Web程序员成功之路

ASP.NET Web



- 迅速提高读者Web开发能力，全面挖掘读者开发潜力
- 一线资深Web程序员经验力作，窗内网独家推荐自学教材
- 17个小时视频教学，简化学习过程
- 101个实战案例与理论知识综合讲解，提高应用能力
- 网站互动教学（www.itzcn.com），QQ群在线帮助读者解疑

韩 啸 王瑞敬 刘健南 编著



随书附赠
DVD-ROM

清华大学出版社

Web 程序员成功之路

ASP.NET Web开发学习实录

韩　啸　王瑞敬　刘健南　编著

清华大学出版社
北　京

内 容 简 介

ASP.NET 是微软力推的 Web 开发编程技术，也是当今最热门的 Web 开发手段之一。ASP.NET 3.5 技术提高了网络系统平台开发的效率和安全性，如新增匿名类型、Lambda 表达式和 LINQ、集成 ASP.NET Ajax、增强的数据控件等。

本书重点围绕 Web 开发，结合精选教学视频，全程推演 ASP.NET 3.5 应用开发的整个过程，引导读者深刻理解和掌握以 ASP.NET 从事 Web 应用开发所需要的基本知识和技能。

全书共 15 章，主要内容包括 ASP.NET 必备组件的使用、页面指令和语法、内置对象和类库、访问数据库和文件以及结合 MVC 框架的知识。另外还对如何实现 Ajax、与 Flash 通信、实现绘图以及安全性有详细介绍。

本书可以作为 ASP.NET 3.5 的入门书籍，也可以帮助中级读者提高技能，对高级读者也有一定的启发意义。

本书封面贴有清华大学出版社防伪标签，无标签者不得销售。
版权所有，侵权必究。侵权举报电话：010-62782989　13701121933

图书在版编目(CIP)数据

ASP.NET Web 开发学习实录/韩啸，王瑞敬，刘健南编著. --北京：清华大学出版社，2011.7
(Web 程序员成功之路)
ISBN 978-7-302-25969-5

Ⅰ. ①A…　Ⅱ. ①韩…　②王…　③刘…　Ⅲ. ①网页制作工具，ASP.NET　Ⅳ. ①TP393.092

中国版本图书馆 CIP 数据核字(2011)第 124096 号

责任编辑： 张　瑜　宋延清
装帧设计： 杨玉兰
责任校对： 周剑云
责任印制：

出版发行： 清华大学出版社　　**地　址：** 北京清华大学学研大厦 A 座
http://www.tup.com.cn　　**邮　编：** 100084
社 总 机： 010-62770175　　**邮　购：** 010-62786544
投稿与读者服务： 010-62776969, c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015, zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　销： 全国新华书店
开　本： 190×260　**印　张：** 43.5　**字　数：** 1051 千字
附 DVD 1 张
版　次： 2011 年 7 月第 1 版　**印　次：** 2011 年 7 月第 1 次印刷
印　数： 1~4000
定　价： 88.00 元

产品编号：

前　言

随着互联网的不断发展和平台的多样化发展，人们越来越依靠个人网站或企业门户网站来实现各种各样的业务和价值了。而在创建网站的各种技术中，ASP.NET 起着不可估量的作用，ASP.NET 是一个统一的 Web 平台，可提供生成企业级应用程序所必需的所有服务。ASP.NET 在.NET Framework 基础上运行，因此所有.NET Framework 功能都适用于 ASP.NET 应用程序。特别是 ASP.NET 3.5 技术提高了网络系统平台开发的效率和安全性，如新增匿名类型、Lambda 表达式和 LINQ、集成 ASP.NET Ajax、增强的数据控件等。

Visual Studio 2008 和 C#是开发 ASP.NET Web 应用程序的最佳搭档，同时这对“组合”也深受广大开发人员的青睐。

本书内容

第 1 章　构建 ASP.NET 平台，安装和配置 ASP.NET 的工作环境和开发环境。介绍 Web 服务器 IIS 的安装和配置、ASP.NET 开发环境 Visual Studio 2008 的安装。使用 Visual Studio 2008 创建一个 Web 项目，然后发布到 Web 服务器上。

第 2 章　介绍 HTML 控件和 ASP.NET 服务器端控件的用法。

第 3 章　简单介绍 ASP.NET 项目文件的结构，介绍 ASP.NET 页面指令及用法，介绍服务器辅助对象的用法和 ASP.NET 页面的生命周期。

第 4 章　介绍在 ASP.NET 中常用的一些程序基础知识点，比如数组、集合、日期、字符串等类的用法，以及数据类型转换和泛型的使用。

第 5 章　介绍 ADO.NET 与当前主流的几种数据库的连接方式，然后以 SQL Server 数据库为例，对数据库进行添加、查询、修改等操作，接下来介绍一个 ORM 框架 Linq to SQL 的用法，最后介绍使用序列化方式存储数据的方法。

第 6 章　介绍 Repeater 控件和 DataList 控件和 GridView 控件数据列表展示、修改和删除功能，然后使用 DetailsView 控件实现单条数据的展示和修改功能，最后介绍 FormView 控件的用法。

第 7 章　介绍 Ajax 的运行机制和运行原理，然后介绍 ASP.NET 提供的 Ajax 控件来实现 Ajax 功能。

第 8 章　介绍 MVC 模式的设计思想以及 ASP.NET MVC 框架的原理，然后分别介绍 URLRouting、数据呈现字典、HttpHelper、Filter，最后介绍全局异常处理的方法和 ASP.NET MVC 下如何实现 Ajax 的功能。

第 9 章　分别介绍 ASP.NET 与 Flash 和 Flex 等两种华丽的表现层技术的交互方法。

第 10 章　介绍如何利用 GDI+控制页面的颜色和在页面上绘图，如绘制文本、图像、柱状图、饼图的方法。

第 11 章　获取服务器端的秘密。介绍 System.IO 命名空间，以及使用该命名空间下的类对文件和文件夹操作的方法及 ASP.NET 文件的上传功能。

第 12 章　介绍 ASP.NET 的安全机制，性能优化方案和单元测试工具 NUnit 的用法，以及

ASP.NET 应用程序用户身份权限验证方式。

第 13 章　精选了 7 个方面来介绍 ASP.NET 的扩展功能，如实现动态添加控件、天气查询、发送邮件和防止重复登录以及第三方组件等。

第 14 章　介绍企业信息管理系统的系统功能、数据库设计方案，以及新闻、产品、会员、订单等模块的管理和用户界面展示功能的实现方案。

第 15 章　介绍鲜花预订系统的系统架构设计和数据库设计方案，以及用户、鲜花、订单、购物车等功能和管理员后台管理功能的实现。

本书特色

本书大量内容来自真实的 ASP.NET 项目，力求通过实际操作使读者更容易掌握 ASP.NET 的管理操作。本书难度适中，内容由浅入深，实用性强，覆盖面广，条理清晰。

(1)　结构独特

通过“视频教学→基础知识→实例描述→实例应用→运行结果→实例分析”形式将每个知识点与实际应用中的问题相结合。

(2)　形式新颖

用准确的语言总结概念、用直观的图示演示过程、用详细的注释解释代码、用形象的比喻帮助记忆。

(3)　提供文档

将一些非常简单的知识点或者理论性的内容安排在这里。通常这些文档没有具体的实际问题，但是读者又必须了解，如一些概念和术语。

(4)　内容丰富

本书涵盖了实际开发中 ASP.NET 技术所遇到的页面指令、控件编程、页面机制、数据库、MVC 框架、Ajax、文件和安全性等方面的热点问题。

(5)　配合光盘

本书为实例配备了视频教学文件，读者可以通过视频文件更加直观地学习 ASP.NET 3.5 的使用知识。

(6)　网站支持

读者在学习或者工作的过程中，如果遇到实际问题，可以直接登录 www.itzcn.com 与我们取得联系，作者会在第一时间内给予帮助。

(7)　贴心提示

为了便于读者阅读，全书还穿插着一些技巧、提示等小贴士，体例约定如下。

提示：通常是一些贴心的提醒，提供建议或让读者加深印象，或者提供解决问题的方法。

注意：提出学习过程中需要特别注意的一些知识点和内容，或者相关的信息。

技巧：通过简短的文字，指出知识点在应用时的一些小窍门。

读者对象

本书具有知识全面、实例精彩、指导性强的特点，力求以全面的知识性及丰富的实例来指导读者透彻地学习 ASP.NET 3.5 各方面的知识。本书可以作为 ASP.NET 3.5 的入门书籍，也可以帮助中级读者提高技能，对高级读者也有一定的启发意义。

本书适合以下人员阅读和学习：

- ASP.NET 3.5 初学者
- 网站开发人员
- 网站维护人员
- 网页制作爱好者
- 各大中专院校的在校学生和相关授课老师
- 其他 ASP.NET 3.5 从业人员

作者团队

本书主要由韩啸、王瑞敬和刘健南编写，其他参与编写/资料整理/程序开发的人员还有卢江、马春兴、孙晓芳、孙宇霞、王超英、王丹花、王龙才、闫建强等。

由于编者水平有限，书中难免存在不足和疏漏之处，恳请读者批评指正。

目　　录



第 1 章　万事开头难——搭建 ASP.NET平台

内容摘要：

开发 ASP.NET 程序离不开一系列的工具。首先最必需的就是有一台配置不错的电脑，然后装上操作系统，这是最基础的。有了搞软件最必备的工具以后，我们就要研究 ASP.NET 所需的环境了。ASP.NET 是一种用于开发 Web 应用程序的技术，其程序需要有 Web 服务器支持。既然是微软的技术，当然首选微软的服务器程序。

现在用户量最大的服务器系统是微软的 Windows Server 2003，其默认的 Web 服务器程序是 IIS 6。Windows Server 2003 + IIS 6 这个组合应该算是当今使用范围最广的 Web 服务器环境了。它能运行 ASP、PHP、ASP.NET，当然还有一些人使用它配合 Tomcat 来发布 JSP 程序。此组合可以说强大至极！

有了服务器平台，还得有它自己需要的处理程序。ASP.NET 需要有.NET Framework 的支持，.NET Framework 现在有 1.1、2.0、3.0、3.5、4.0 等版本。3.x 版本的.NET Framework 都是以 2.0 的一些扩展包的方式出现的。所以我们现在最常用的.NET Framework 3.5 实际对应的版本是 2.0.50727。

有了 ASP.NET 的运行环境，还需要建立开发环境。当然是选择微软的 Visual Studio(VS)。虽然 VS2010 已经发布，但现在使用最多的还是 VS2008。

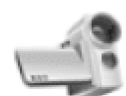
这一章就来学习一下 ASP.NET 运行环境 IIS 6 + .NET Framework 3.5 的搭建和配置，以及开发环境 VS2008 的安装和调试。

学习目标：

- 掌握 IIS 的安装和常用配置
- 掌握.NET Framework 的安装及 IIS + .NET Framework 的配置
- 掌握 ASP.NET 开发环境(VS2008)的安装
- 了解.NET 应用程序的编译运行原理
- 熟练使用 VS2008 创建和发布应用程序

1.1 安装和配置ASP.NET服务器

前面提到，现在网络上应用最多的 Web 应用程序服务器是 Windows Server 2003 + IIS 6，ASP.NET 应用程序可以运行在 IIS 6 上，所以我们可以在 Windows Server 2003 中配置自己的 Web 服务器 IIS 6。

**视频教学：**光盘/videos/1/InstallIIS.avi　　　　长度：5 分钟

1.1.1 基础知识——IIS

IIS(Internet Information Services，互联网信息服务)是由微软公司提供的基于 Windows 操作系统的互联网基础服务。最初是 Windows NT 版本的可选包，随后内置在 Windows 2000、Windows XP Professional 和 Windows Server 2003 中一起发行。

IIS 其实是一种 Web 服务组件，其中包括 Web 服务器、FTP 服务器、NNTP 服务器和 SMTP 服务器。它提供 ISAPI(Intranet Server API)，作为扩展 Web 服务器功能的编程接口。

IIS 是发布网站最为常用的工具，通过 IIS，可以把网站发布到 Internet 上，使 Internet 上的其他用户可以访问该网站。

1.1.2 实例描述

前面我们曾说过，我们的 ASP.NET 应用程序最终要运行在 Windows Server 2003 + IIS 6 + .NET Framework 上。

“工欲善其事，必先利其器。”所以在这一节来学习 Windows Server 2003 下如何安装和配置 IIS 6 以及.NET Framework。

1.1.3 实例应用

【例 1-1】安装和配置 ASP.NET 服务器。

要配置 ASP.NET Web 服务器，需要在 Windows Server 2003 服务器系统上安装 IIS 6、.NET Framework 3.5，还要运行 ASP.NET 应用程序。



虽然.NET Framework 最新版本是 4.0，但是应用最广的还是.NET Framework 3.5。所以这里使用 3.5 版本的.NET Framework。

1. 安装IIS 6

我们先来在服务器(Windows Server 2003)上安装 IIS 6。在安装之前，需要先把 Windows Server 2003 系统安装盘放入光驱，或准备一份 IIS 6 安装包，解压备用。



Web 服务器一般是在 Internet 上发布的具有独立 IP 的高性能主机。我们这里只是测试，所以用虚拟机虚拟一台 Windows Server 2003 服务器。

(1) 运行 Windows Server 2003 系统。依次单击“开始”→“设置”→“控制面板”菜单命令，打开“控制面板”窗口。选择“添加或删除程序”图标，如图 1-1 所示。

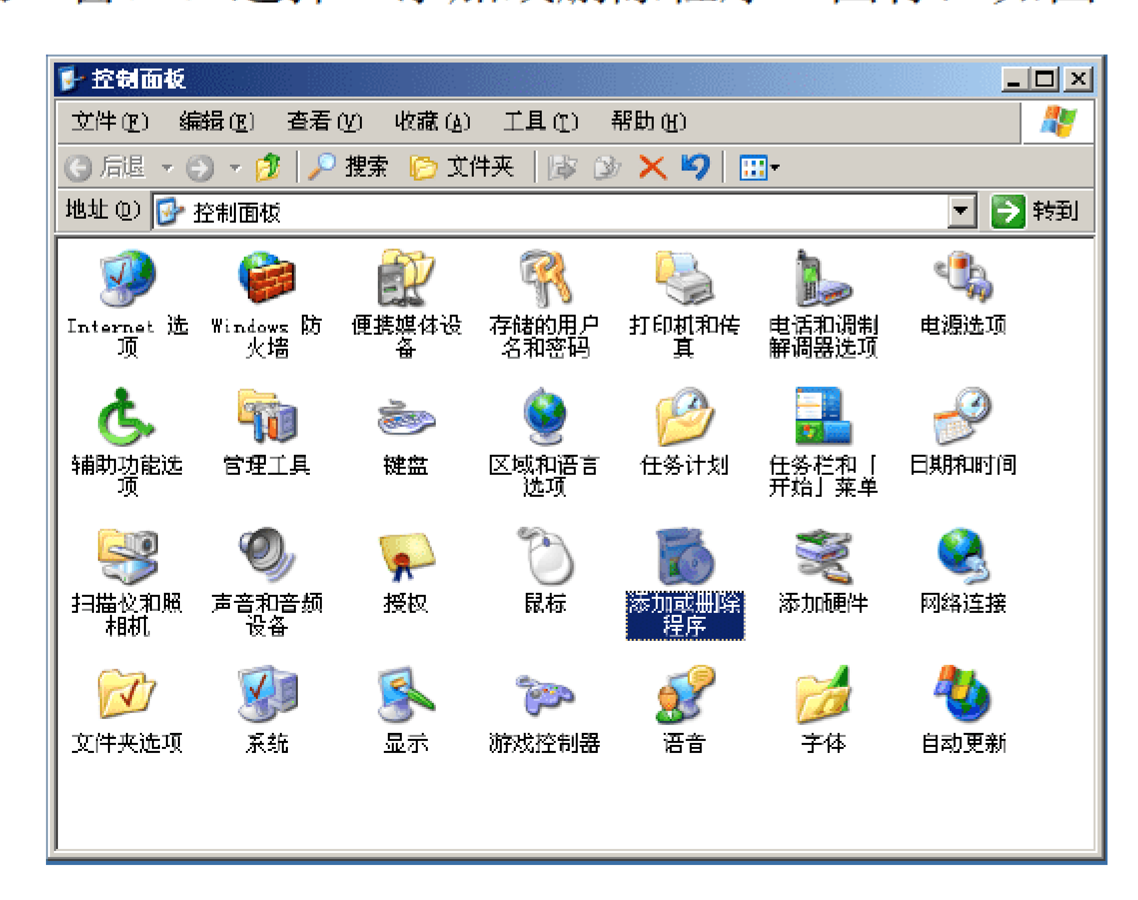


图 1-1　“控制面板”窗口

(2) 双击“添加或删除程序”图标，打开“添加或删除程序”对话框。选择左侧的“添加/删除 Windows 组件”图标，弹出“Windows 组件向导”对话框，在“组件”列表框中选择“应用程序服务器”选项，如图 1-2 所示。

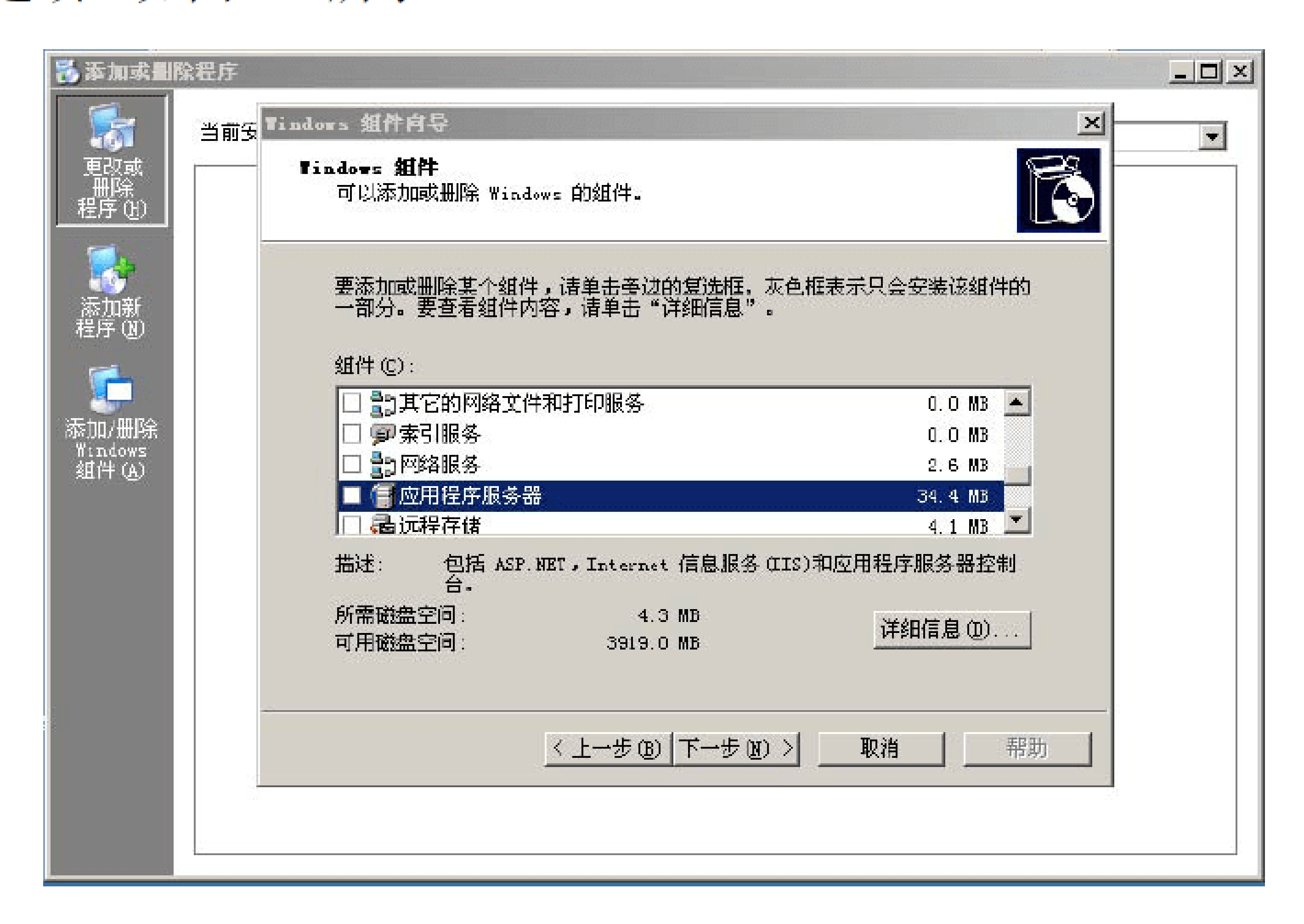


图 1-2　Windows组件向导

(3) 选中“应用程序服务器”选项以后，单击“详细信息”按钮，打开“应用程序服务器”对话框，选中“应用程序服务器的子组件”列表框中的“Internet 信息服务(IIS)”选项，系统会自动选中“启用网络 COM+访问”，不管它。我们需要运行 ASP.NET 程序，所以这里选中 ASP.NET 项，如图 1-3 所示。

(4) 单击“确定”按钮关闭“应用程序服务器”对话框。返回“Windows 组件向导”对话框。单击“下一步”按钮开始安装，如图 1-4 所示。

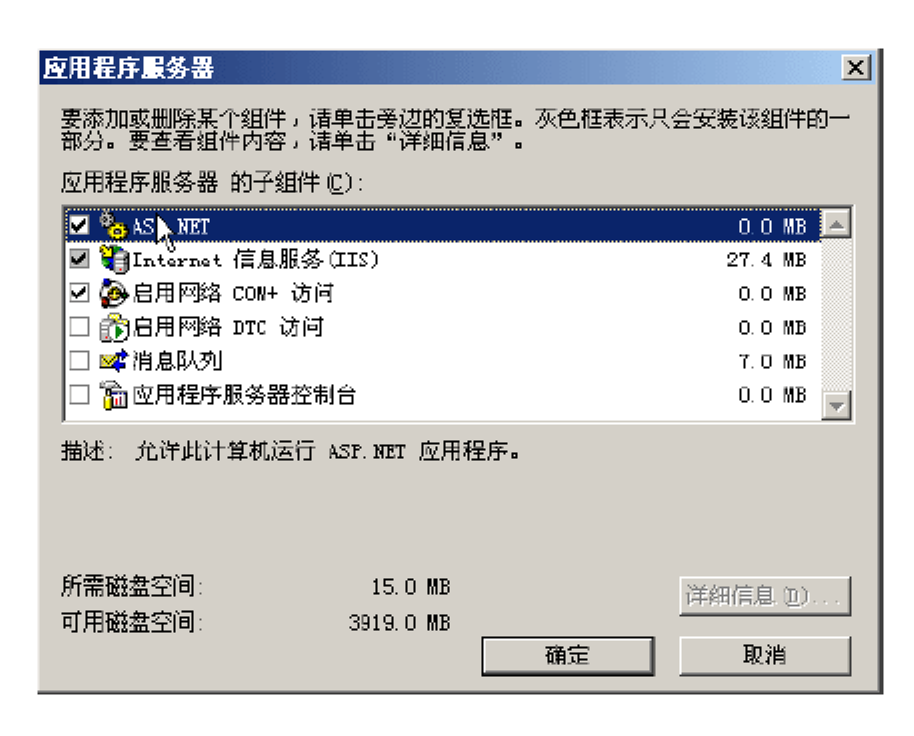


图 1-3 “应用程序服务器”对话框

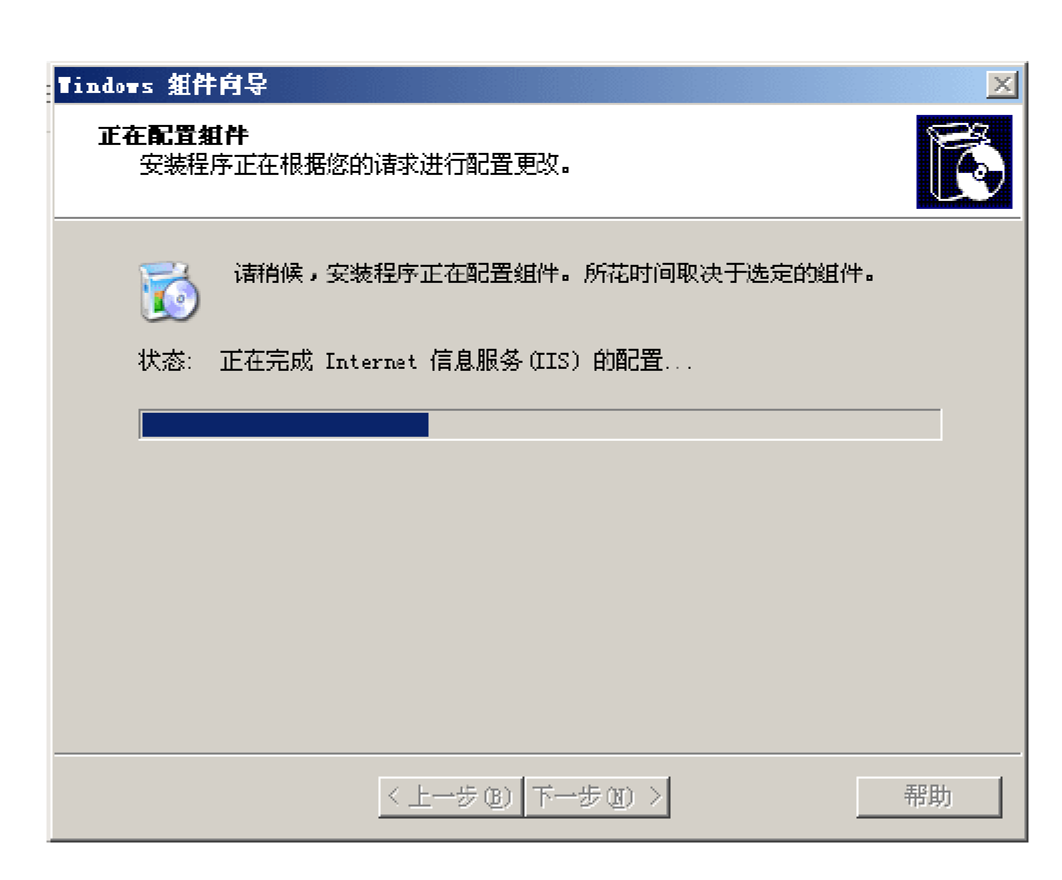


图 1-4 Windows组件安装过程

(5) 在安装过程中可能会因为你准备的安装程序目录和安装系统时的系统盘目录不一致而导致系统不能自动找到安装文件，弹出“所需文件”对话框。如图 1-5 所示。单击“浏览”按钮，找到我们安装光盘里的 I386 目录或者 IIS 安装包解压目录，选择相应的文件，单击“确定”按钮即可，如图 1-6 所示。如有需要，重复该步骤。



图 1-5 “所需文件”对话框

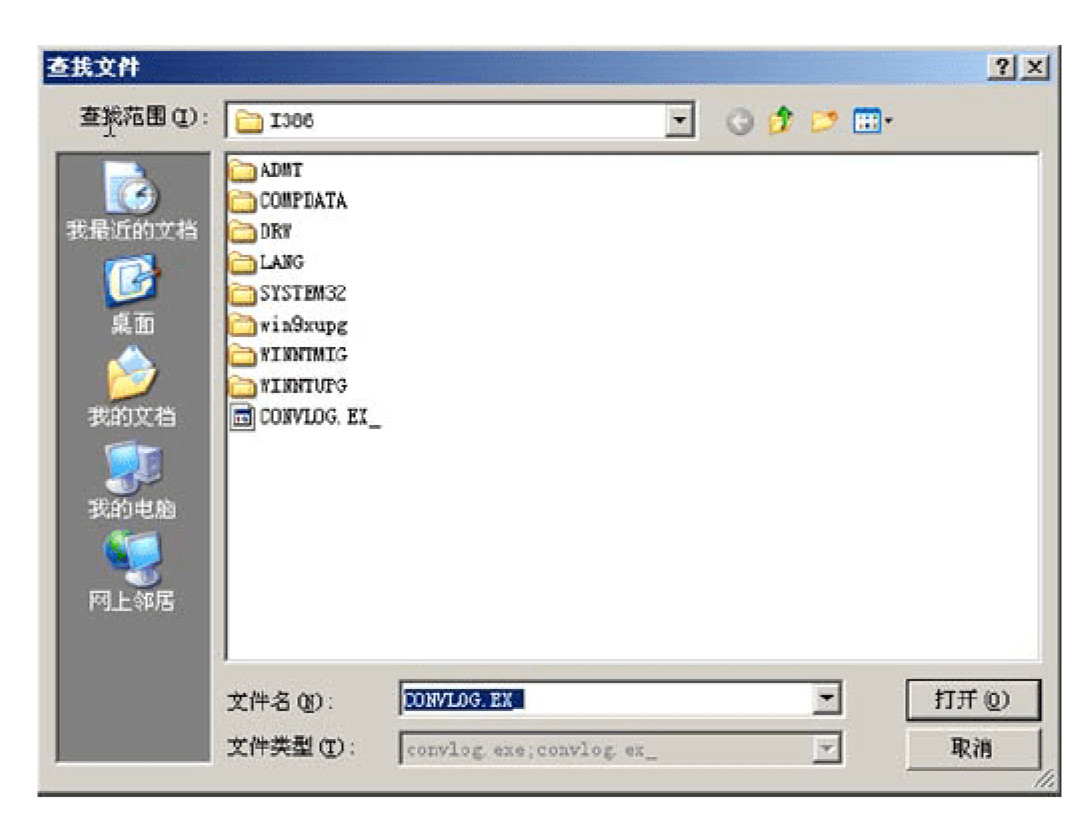


图 1-6 “查找文件”对话框

(6) 稍等片刻，即可完成安装，如图 1-7 所示。



图 1-7 完成“Windows组件向导”界面

至此，IIS 6 安装就完成了。

2. 安装.NET Framework 3.5 支持

要支持 ASP.NET Framework 3.5 的应用程序，当然需要安装.NET Framework 3.5 了。.NET Framework 3.5 非常庞大，网上下载的.NET Framework 3.5 一般超过 200MB。下面是安装过程。

(1) 双击运行安装程序，它会先将安装程序解压到硬盘的某个分区，再执行安装过程。稍等片刻，便会出现如图 1-8 所示的使用协议的界面。



图 1-8　“欢迎使用安装程序”界面

(2) 选中“我已经阅读并接受许可协议中的条款”单选按钮，单击“安装”按钮。来到“下载和安装进度”界面，开始下载所需要的安装文件，如图 1-9 所示。根据个人所在网络的网速不同，所需要的时间也不尽相同。等下载完成，会自动进入安装界面，安装成功以后弹出安装完成界面，如图 1-10 所示。

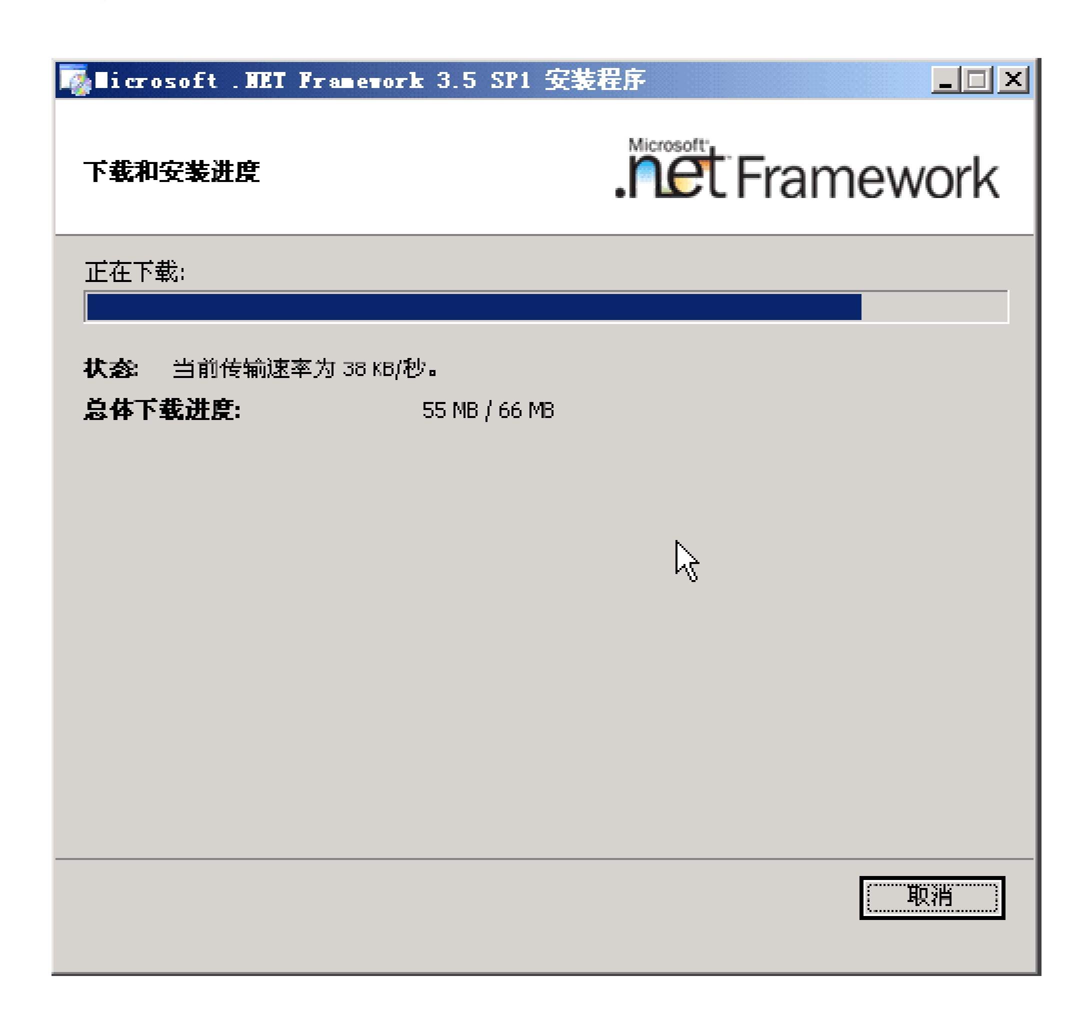


图 1-9　下载和安装进度

图 1-10　安装完成

(3) 单击“退出”按钮结束安装过程。

1.1.4 运行结果

在浏览器的地址栏中输入“http://localhost”来访问 Web 服务器，若看到如图 1-11 所示的界面，就说明 IIS 已经安装成功。

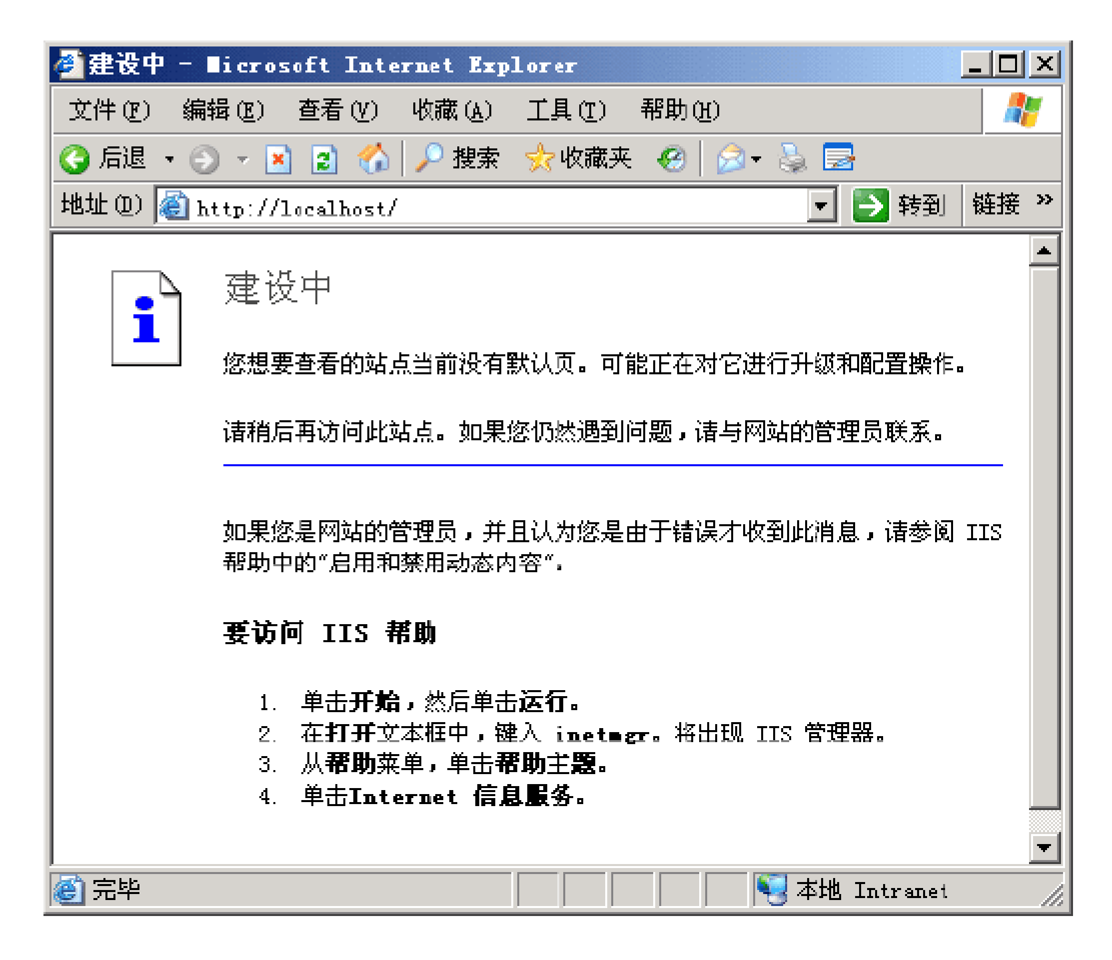


图 1-11 访问服务器的请求结果

然后打开“控制面板”里的“管理工具”，运行“Internet 信息服务(IIS)管理器”。展开左侧“(本地计算机)”节点和其下的“网站”节点。右键单击“默认网站”节点，从弹出的快捷菜单中选择“属性”命令，打开“默认网站 属性”对话框，切换到 ASP.NET 选项卡，如图 1-12 所示，说明.NET Framework 3.5 也已经安装成功。



图 1-12 ASP.NET选项卡

这里我们把“ASP.NET 版本”选择为“2.0.50727”后，单击“确定”按钮。设置默认网站支持的 ASP.NET 版本为.NET Framework 3.5。

1.2　安装ASP.NET开发工具

前面我们已经安装配置了 ASP.NET 的运行环境。

但“巧妇难为无米之炊”，没程序我们运行什么呢？

程序从何而来？当然需要我们来开发，本书就是为了学习软件开发的。

开发就得有工具。好的开发工具能减少很多的劳动。我们开发 ASP.NET 应用程序，就需要使用微软设计的一个非常强大的集成开发环境——Visual Studio，简称 VS。

这里我们使用 VS2008，因为它是当下最为流行的开发工具。我们一起来学习 VS2008 的安装过程。



这里并不是在服务器上安装 VS2008，而是在我们本地计算机的 Windows XP 系统上安装。



视频教学：光盘/videos/1/InstallVS.avi　　　　长度：5 分钟

1.2.1　实例描述

话说有两个小孩去砍柴，一个小孩想：我肯定得赶紧干活，才能比他砍得多；另一个小孩看斧子有点锈了，就跑去磨斧子了。

不多久磨好了斧子，歇了一会才去砍柴。到那里一看，第一个小孩累死累活的才砍了没几根，磨斧子的这个小孩提起斧子，几下就超过了他，第一个小孩后悔的不得了……

这其实就是“磨刀不误砍柴工”的故事。

用这个故事来比喻搞好工具的重要性。下面我们来安装集成开发环境 VS2008。

1.2.2　实例应用

【例 1-2】安装 ASP.NET 开发工具 Visual Studio 2008。

(1)　我们先把买来的安装光盘放入光驱，或将下载的安装文件载入虚拟光驱。双击光驱盘符运行安装程序，如图 1-13 所示。

(2)　单击“安装 Visual Studio 2008”链接，打开 Visual Studio 2008 正在加载安装文件的对话框，如图 1-14 所示。

(3)　加载完成以后，进入欢迎界面，单击“下一步”按钮，进入“起始页”界面，如图 1-15 所示。针对许可协议，我们肯定要选中“我已阅读并接受许可条款”单选按钮，不接受就安装不下去了。然后下面有产品密钥，作者使用的版本可以试用三个月，所以这里不用输入密钥。直接单击“下一步”按钮。进入“选项页”。

(4)　在“选项页”中选择要安装的功能。我们选择“自定义”。“产品安装路径”选择默认路径，当然你可以选择其他目录，如图 1-16 所示。



图 1-13　运行Visual Studio 2008 安装程序

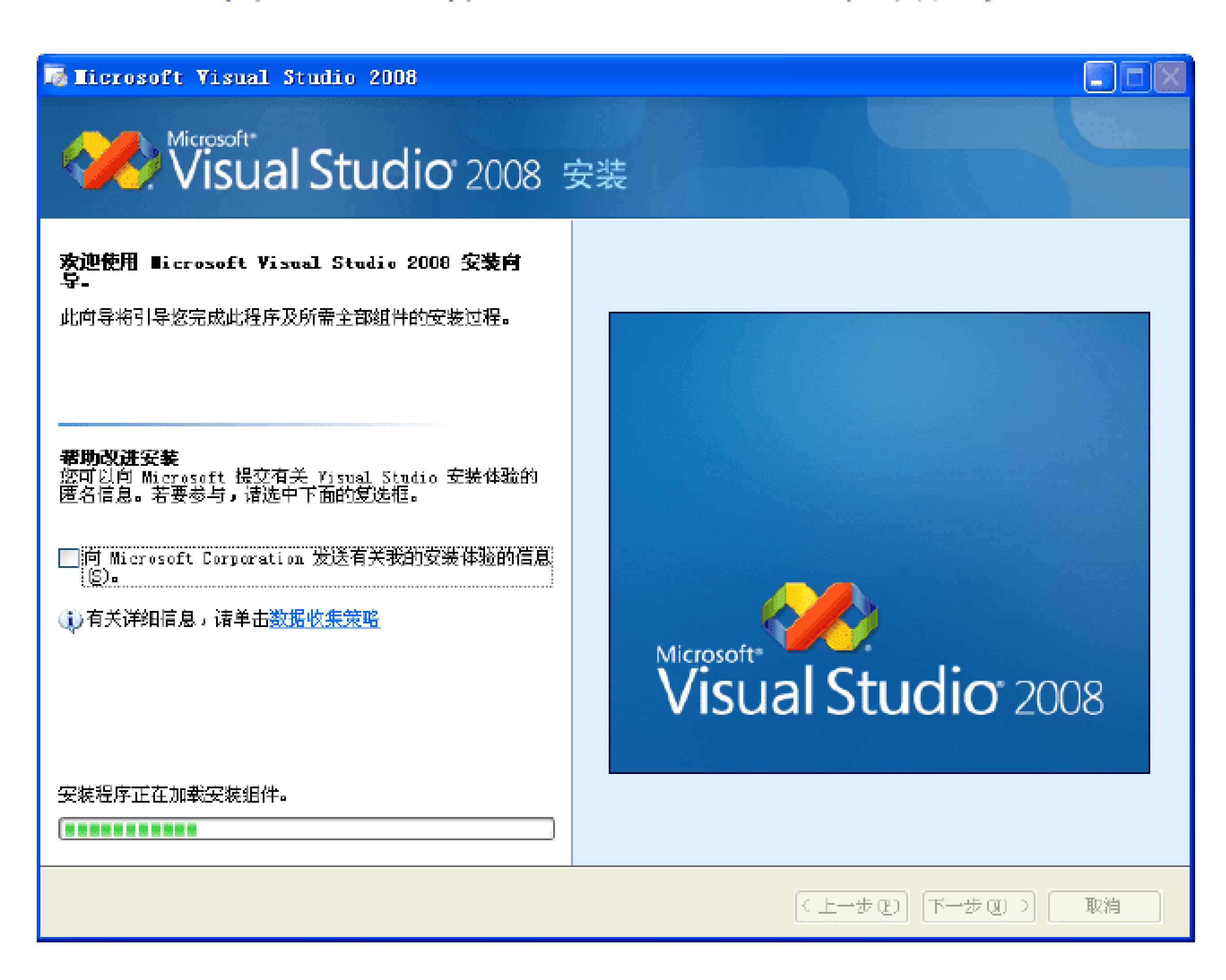


图 1-14　Visual Studio 2008 正在加载安装文件

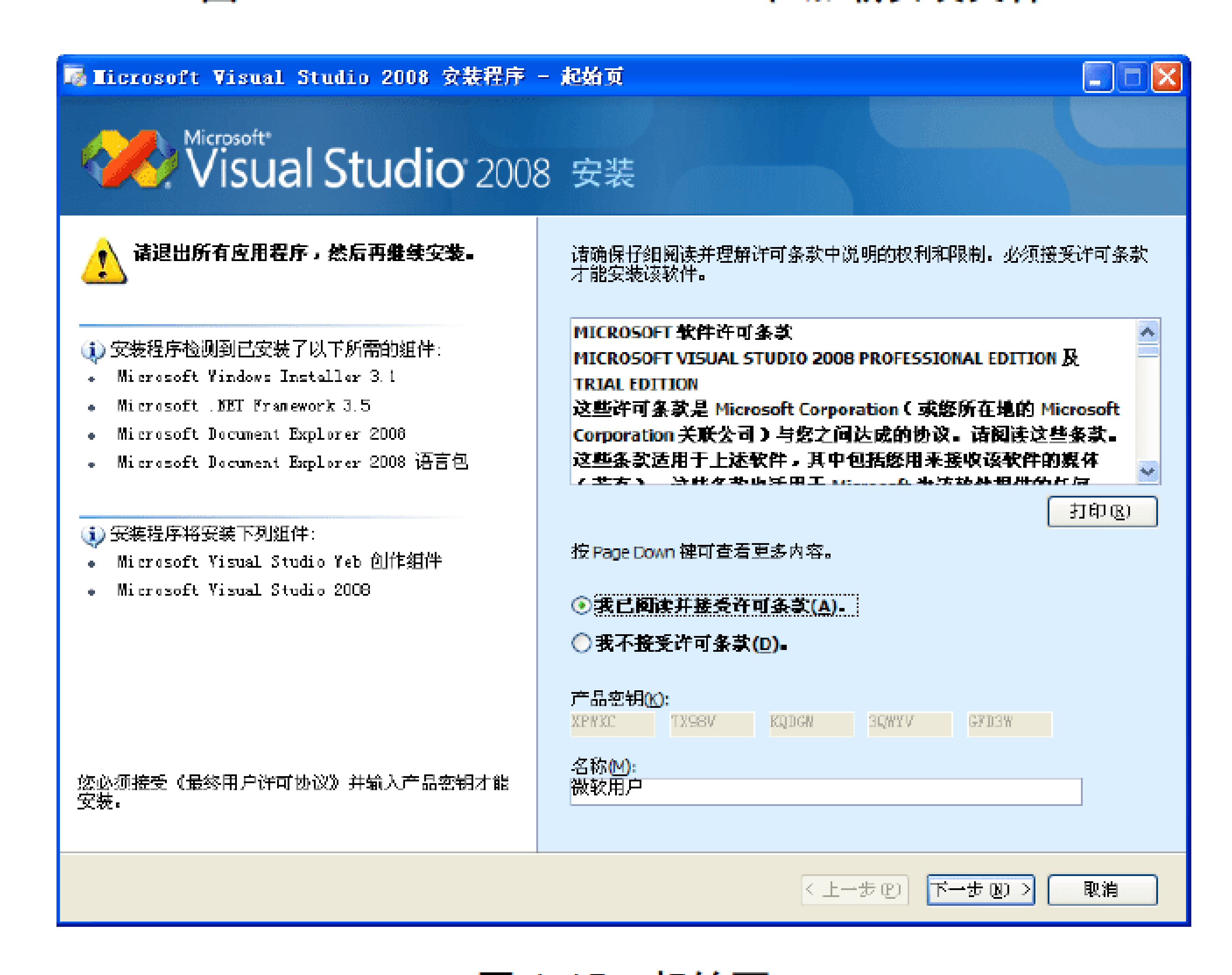


图 1-15　起始页



图 1-16　选项页

(5) 单击“下一步”按钮。选择你要安装的功能，把不需要安装的选项去掉，这里只使用C#语言，所以去掉了“语言工具”下面的“Visual C++”和“Visual Basic”前面的对勾。看看“所需磁盘空间”立刻减少一个多“GB”，如图 1-17 所示。所以我们应挑选自己根本用不到的组件，取消对它们的安装。



图 1-17　只选中你要安装的功能

(6) 单击“安装”按钮，开始安装，如图 1-18 所示。这里为了演示，只选择了部分组件，所以安装列表里只有作者选择的那几个组件，正常的完全安装会有几十个组件要安装。一般安装过程会持续几十分钟。

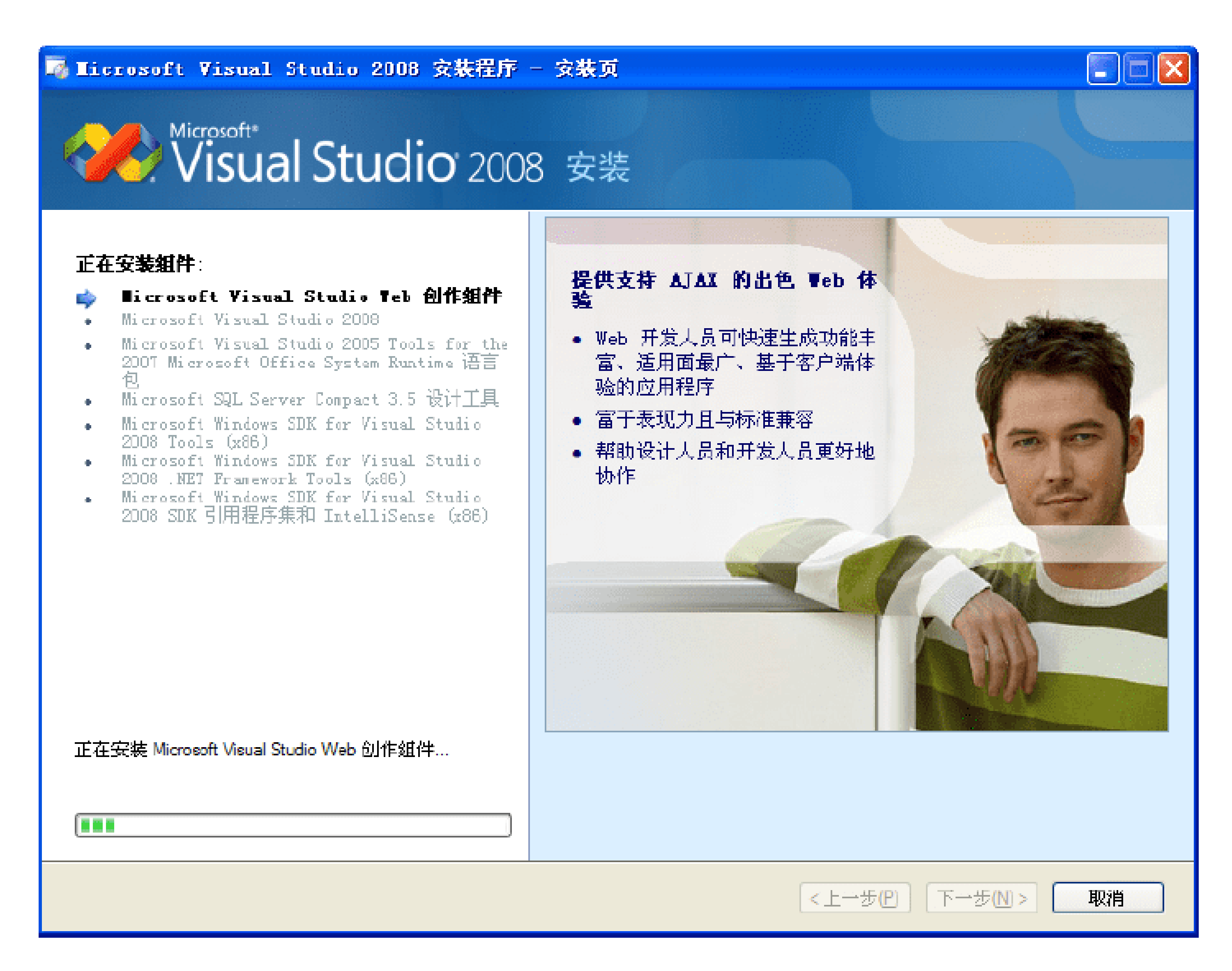


图 1-18 正在安装VS2008

(7) 不多久就会提示安装成功，如图 1-19 所示。单击“完成”按钮结束安装。



图 1-19 安装完成

1.2.3 运行结果

安装完成后运行 Microsoft Visual Studio 2008。因为是第一次运行，所以弹出“选择默认环境设置”对话框，如图 1-20 所示。

这里选择“Web 开发设置”，单击“启动 Visual Studio”按钮。系统会自动配置一下环境，可能需要几十秒钟。完成后就打开 VS 进入 Microsoft Visual Studio 起始页。单击“帮助”菜

下的“关于 Microsoft Visual Studio”，可以打开“关于 Microsoft Visual Studio”对话框，如图 1-21 所示。

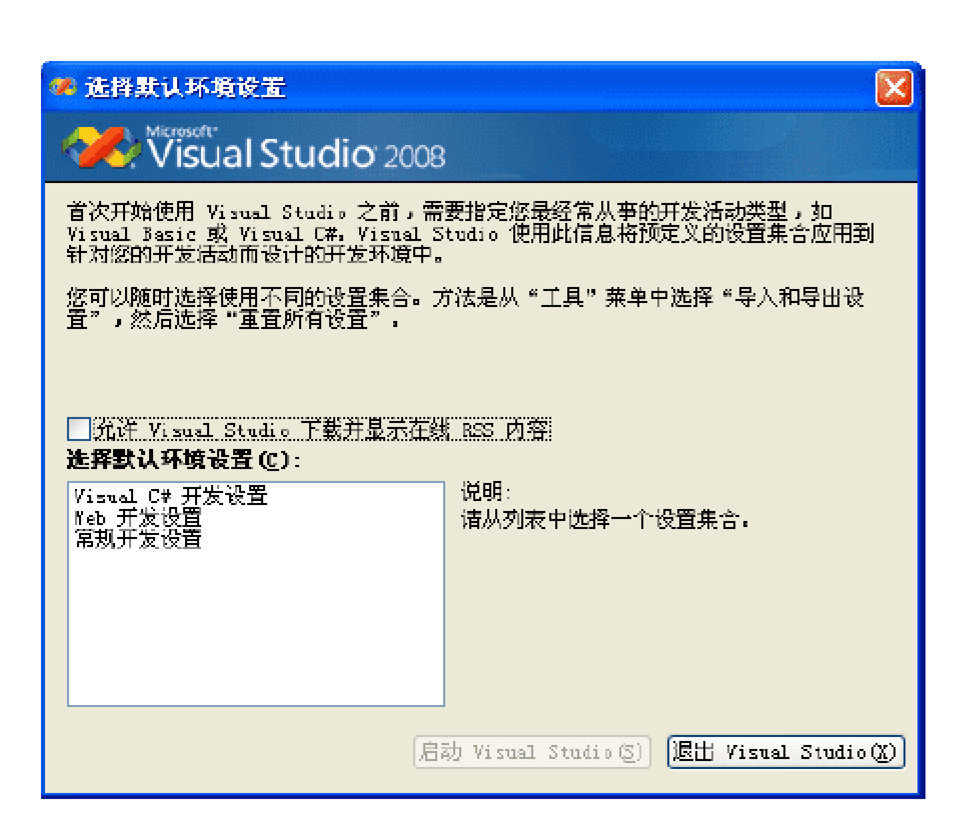


图 1-20　“选择默认环境设置”对话框

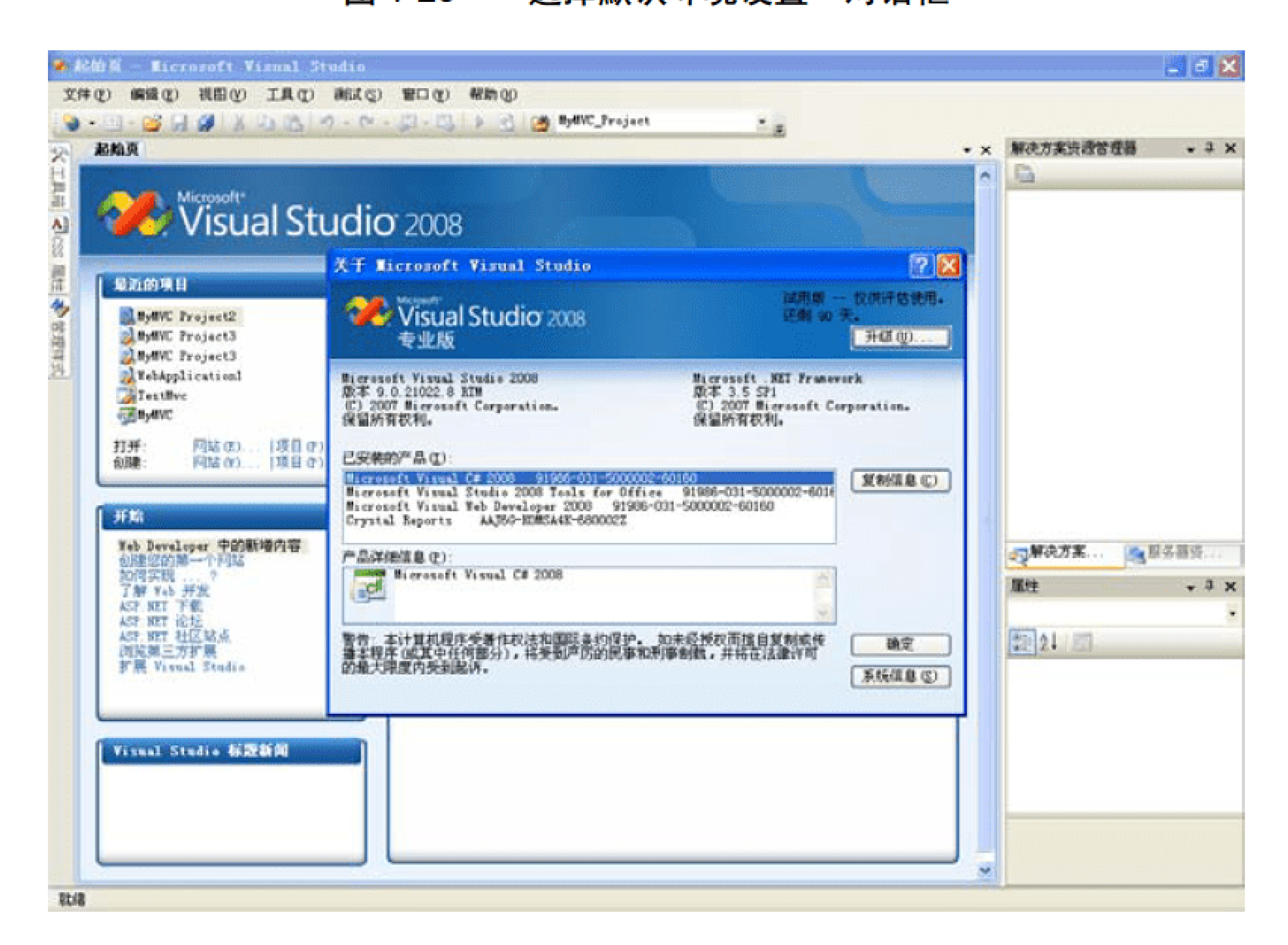


图 1-21　“关于Microsoft Visual Studio”对话框

至此，说明 VS2008 中文版已经安装成功。

1.3　使用记事本开发简单的计算器

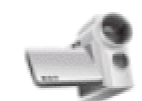
VS 确实很强大。但是它也太强大了，强大到我们可能不知道它都干了些什么。

每次我们只需要打开它、创建项目、新建类，写完代码后运行就可以了。但我们实际做了些什么呢？有点不太清楚。

毕竟 VS 也只是个工具，工具只是为了帮我们干活的。假如现在只是为使用工具而使用工具，连我们真正要做什么都不清楚了，那似乎是一种悲哀吧！

如果不知道自己在干什么，心里总会是空落落的。没关系，这里我们先不用工具，我们用记事本写一个小程序，手动编译、执行一下试试。

别小看这节啊。只有吃尽苦头，才会明白幸福二字的含义。同样你只有深刻地体会了不用IDE(集成开发环境)的痛苦，才能体会出使用 IDE 的幸福。


视频教学：光盘/videos/1/csharp.avi　　长度：8 分钟

1.3.1　基础知识——手动编译C#类

C#程序源码是文本文件(其实高级语言的源码都是文本文件)。所以，我们可以用记事本来编写它。而且 C#编译器对文本文件的扩展名也没有要求，不像 Java 那样扩展名必须是.java。

C#的扩展名可以是任意字符，不过官方建议是以.cs 结尾，而且用 VS 创建的 C#类扩展名默认都是.cs。

C#程序最基本的代码块的单位是“类”。也就是说只要是一段完整的代码，就必须至少有一个类将它们包含起来。

程序一般都有一个入口点。C#程序的入口点是一个名为 Main 的方法。其实很多程序的入口点都是这个名称的方法，比如 Java 程序和 C 程序。

C#程序的 Main 方法在一个程序里只能出现一次；必须是静态方法；返回值只能是 int 或者 void；参数可以不加，但如果加参数，就有且只能有一个 string[]类型的参数；并且方法名必须以大写字母开头，其他字母小写。

当然，Main 方法也必须放在一个类里。

用 C#编译源程序时，需使用.NET Framework 目录里的 csc.exe。作者机器上的 csc.exe 文件存放在 C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727 目录下。

为了手工编译方便，把这个路径加入系统环境变量里。具体操作为：右击桌面上“我的电脑”图标，选“属性”菜单命令，打开“系统属性”对话框，选“高级”选项卡，单击下面的“环境变量”按钮，打开“环境变量”对话框，编辑下面“系统变量”列表里的 Path 变量值，在最后面追加一个半角的分号“;”和 csc.exe 文件所在目录的路径。



如果还不怎么明白这个过程，或觉得说得有点简略，可以百度一下“如何添加环境变量”，有一大堆方法可以参考。

这样我们就可以直接在任一目录使用 csc.exe 来编译 C#源码而不用加很长的目录了。

1.3.2　实例描述

既然本节我们是想研究 C#程序的手工编译运行，就举一个简单的例子来演示一下手工编译执行控制台程序的整个步骤吧。

最早时候，计算机的发明是用来进行数据计算的。所以我们也不妨复古一下，用记事本写一个简单的二维计算器来演示这种例子。

闲言少叙，现在就开始进入正题。

1.3.3　实例应用

【例 1-3】使用记事本开发简单的计算器。

首先在工作目录下创建一个文本文件，这里命名为 Calculator.txt。这样命名只是为了帮助理解后缀的无关性，建议读者仍用.cs 作为源程序扩展名。

创建完以后打开该文本文件。

必须先引入命名空间。这里只需要 System 就可以了：

```
using System;
```

再创建一个 Counter 类。该类有 4 个方法，分别执行加、减、乘、除操作。代码如下：

```
class Counter
{
    //执行加法计算
    public decimal Add(decimal value1, decimal value2)
    {
        return value1 + value2;
    }

    //执行减法计算
    public decimal Subtract(decimal value1, decimal value2)
    {
        return value1 - value2;
    }

    //执行乘法计算
    public decimal Multiply(decimal value1, decimal value2)
    {
        return value1 * value2;
    }

    //执行除法计算
    public decimal Divide(decimal value1, decimal value2)
    {
        return value1 / value2;
    }
}
```

程序比较简单，就不做解释了。

我们再来创建一个 Calculator 类。该类包含 Main 方法，调用 Counter 类执行计算操作。代码如下：

```
class Calculator {
    static void Main(string []args)
    {
        decimal value1 = decimal.Parse(args[0]);    //数值 1
```

```
        string op = args[1]; //运算符
        decimal value2 = decimal.Parse(args[2]); //数值 2
        decimal result = 0;  //运算结果

        Counter counter = new Counter(); //声明一个实例变量

        switch(op)    //根据运算符选择相应的方法执行运算
        {
            case "+":
                result = counter.Add(value1, value2);
                break;
            case "-":
                result = counter.Subtract(value1, value2);
                break;
            case "*":
                result = counter.Multiply(value1, value2);
                break;
            case "/":
                result = counter.Divide(value1, value2);
                break;
        }

        Console.WriteLine(value1 + op + value2 + "=" + result); //打印结果
    }
}
```

由于很简单，这里也不做解释了。

我们来编译运行一下。

1.3.4 运行结果

因为是控制台程序，所以要在控制台下运行。

单击“开始”菜单，选择“运行”命令，打开“运行”对话框，输入“cmd”，单击“确定”按钮，即可打开控制台窗口。

在控制台窗口中找到你的工作目录，这里是在 E:\Zhh\Project\Calculator 下，使用命令“e:”切换到 E 盘下，再用“cd E:\Zhh\Project\Calculator”切换到工作目录下。

执行命令“csc calculator.txt”来编译我们的源程序。之后，得到了一个 Calculator.exe 文件。扩展名是.exe，当然是可执行文件了。

我们来执行它。当然，需要提供三个参数：数值 1，运算符，数值 2。

例如：

```
calculator 1 + 1
```

当然，它们每一项之间的空格是不能少的。

我们来看一下执行结果，如图 1-22 所示。

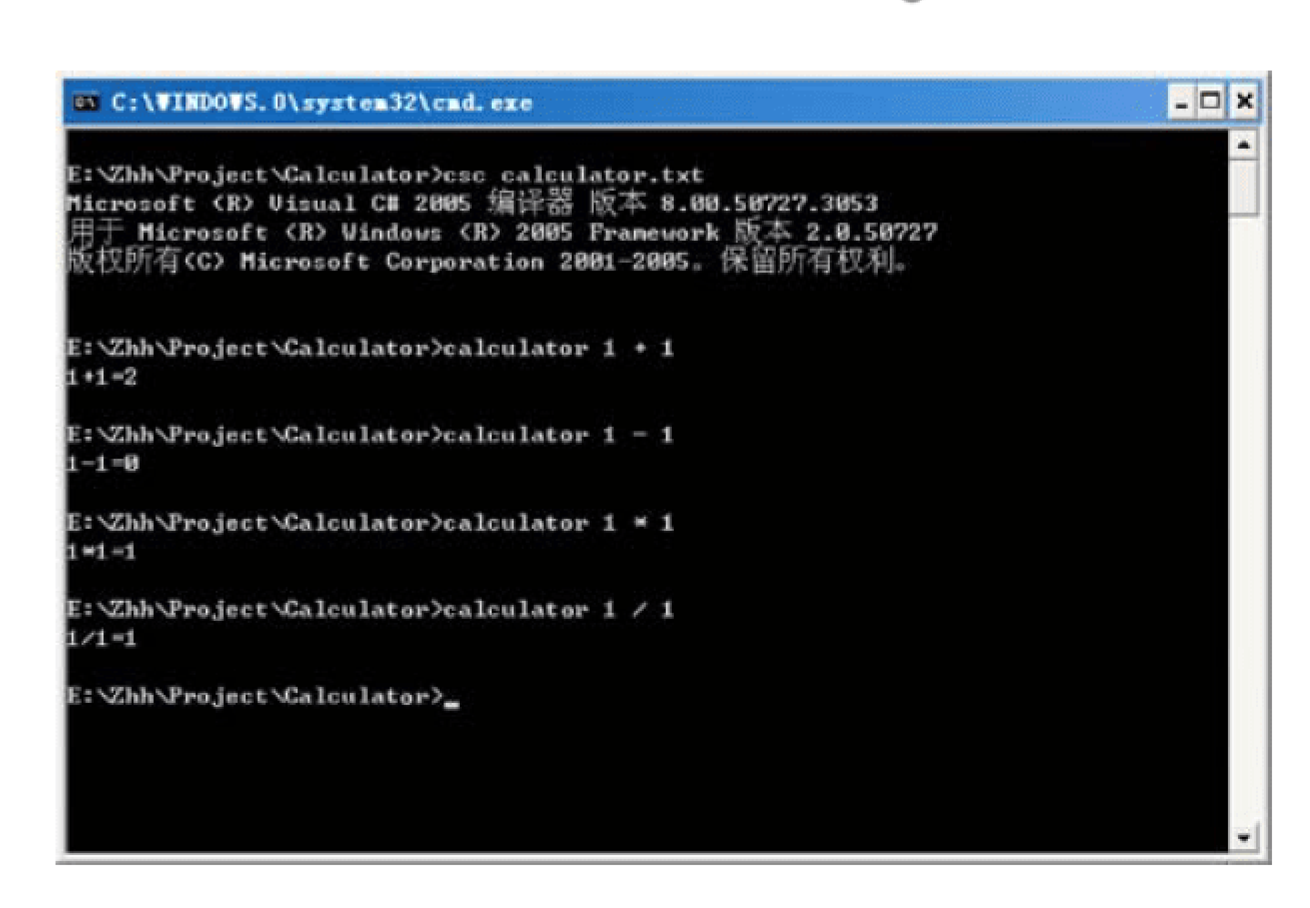


图 1-22　程序在控制台窗口中的运行结果

1.3.5　实例分析

源码解析：

其实费了这么多笔墨，主要还是围绕一个 csc 编译命令来讲解 C#应用程序的编写、编译、运行过程。csc 命令有很多种用法，我们在控制台输入“csc -?”就可以查看 csc 命令的帮助文档。这里我们没有特殊要求，只在命令后面跟一个要编译的源代码文件的文件名，即可编译生成该程序的可执行文件。

1.4　创建一个简单的用户登录

上节我们了解了不用开发工具来创建 C#应用程序的方法。本节我们来使用强大的集成开发环境 VS2008 来创建一个简单的 Web 程序。

也许有人会纳闷：刚才还是控制台程序，现在就搞 Web 程序了。是不是有点太跨越了呢？其实，使用 VS 时，实现这种跨越是非常简单的。

视频教学：光盘/videos/1/FirstWeb.avi　　　　长度：4 分钟

1.4.1　基础知识——简单了解服务器端控件

ASP.NET 提供了一套强大的服务器端控件，封装了 HTML 表单元素。可以像 WinForm 程序那样通过拖曳控件来开发页面视图。

本节我们要用到三个服务器端控件：Label、TextBox、Button。

- Label：该控件能很方便地向页面输出一些文本。
- TextBox：该控件就是一个文本框，它用来获取用户的输入。

● Button：该控件是一个命令按钮，可以触发一个服务器端方法，执行相应的操作。

这些控件都能像 HTML 元素一样对它们设置字体、背景色、宽、高等样式，当然也可以用 CSS 来控制它们的样式。

既然是像 WinForm 程序一样的服务器端控件，当然也可以像 WinForm 程序一样给它们设置事件处理程序。

ASP.NET 还允许“页面”和“页面事件处理程序”代码分离为两个文件，也就是说把事件处理程序的代码分离到一个单独的代码文件里，页面中则没有任何逻辑。

下面来看一个例子。

1.4.2 实例描述

在几乎所有的应用程序中，登录功能是必不可少的。它是最简单，最直接，也是最方便的权限验证方法。

本节我们就以登录为例，开发一个简单的 ASP.NET 应用程序。

在这个例子里，当用户输入用户名和密码，单击“登录”按钮时，程序将测试所输入的用户名。如果用户名等于“admin”并且密码等于“123456”，就跳转到登录成功页面，否则提示用户名或密码错误。

1.4.3 实例应用

【例 1-4】创建一个简单的用户登录。

首先我们来创建一个项目。运行 VS2008，选择“文件”→“新建项目”菜单命令。这里的项目取名为“FirstWebProject”，如图 1-23 所示。

创建完以后，我们来看看“解决方案资源管理器”，如图 1-24 所示。这里简单地对默认创建的各项说明一下。

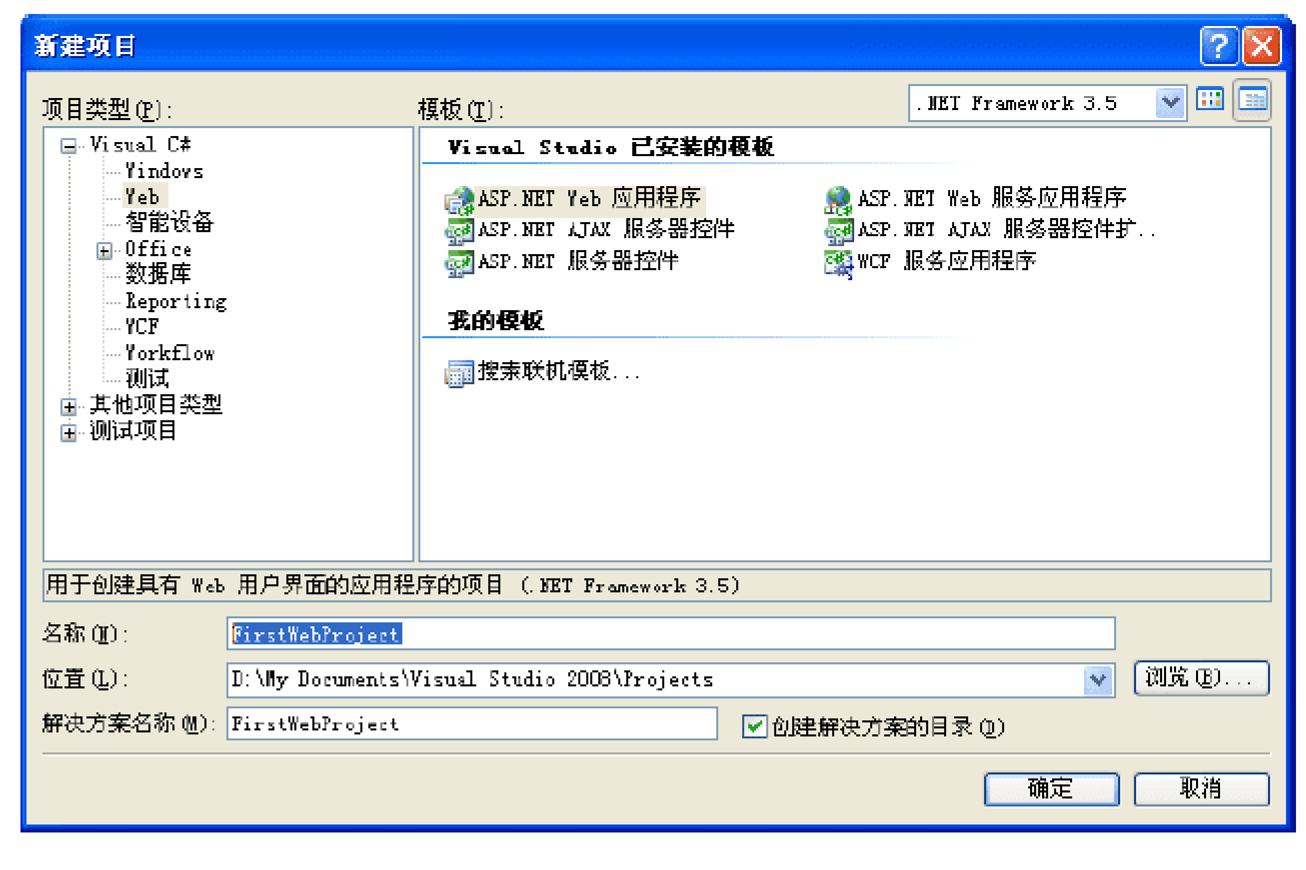


图 1-23 “新建项目”对话框

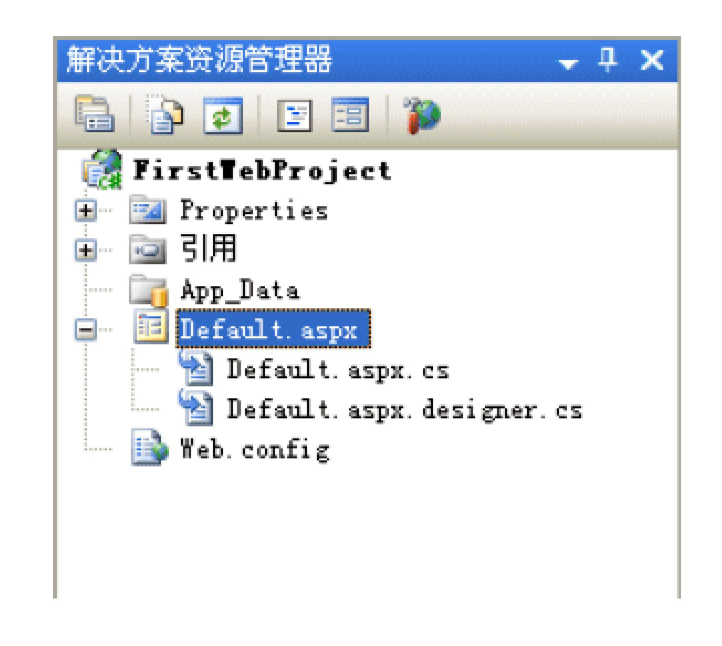


图 1-24 解决方案资源管理器

● Properties：其下存放的是整个项目的属性文件。

- 引用：该目录是管理整个项目所需要的命名空间。
- App_Data：该目录存放了一些系统要用到的、但不想让用户直接访问的资源(比如数据库文件)。
- Default.aspx：这就是默认的页面了，相当于网站首页。其节点下有两个选项。
  - ◆ Default.aspx.cs：让用户编辑的，专门存放事件处理程序和用户声明的方法。
  - ◆ Default.aspx.designer.cs：存放页面服务器控件设计信息。
- Web.config：存放对整个站点的配置信息。

我们双击打开 Default.aspx 文件(其实创建项目的时候它是默认打开的)。看它的界面，比较眼熟吧！与 Dreamweaver 差不多，也分为设计、拆分、源代码三个视图模式。

我们切换到“拆分”视图，这样就可以同时看到设计和代码界面了。

打开窗口左侧的工具箱，在“标准”组里的前三个工具就是刚讲过的三个控件。

这里我们拖一个 Label 控件到页面设计区，为了不影响页面登录时的美观，我们在代码区删掉它的 Text 属性和值。

把光标切换到视图区域中，按 Enter 键换行。输入“用户名：”后，再拖一个 TextBox 控件到页面上。这里不用修改什么。再换行。

输入文本“密　码：”，然后拖入一个 TextBox 控件。这里我们修改它的 TextMode 属性为 Password。

再换行，拖一个 Button 控件到页面，修改其 Text 属性为“登录”。

完工！

这时，页面代码区应该多了如下一段代码：

```
<asp:Label ID="Label1" runat="server"></asp:Label>
<br />
用户名：<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<br />
密　码：<asp:TextBox ID="TextBox2" runat="server" TextMode="Password">
</asp:TextBox>
<br />
<asp:Button ID="Button1" runat="server" Text="登录" />
```

视图区应该如图 1-25 所示。

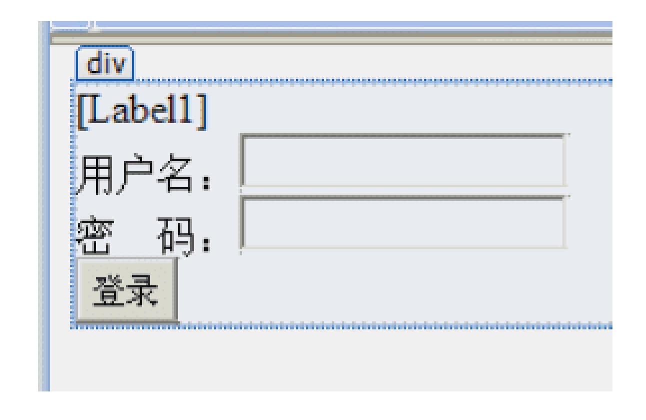


图 1-25　用户登录界面的设计视图

界面设计好之后，我们就可以写代码了。

在视图区双击“登录”按钮，会自动打开 Default.aspx.cs 文件，并在里面创建一个名为 Button1_Click 的方法。该方法就是“登录”按钮的 Click 事件处理方法，在单击该按钮的时候会自动执行方法里的内容。

读者感觉如何？是不是与窗体应用程序一样呢。

接下来我们给事件处理程序加代码，也就是事件处理的代码。

先用 Text 属性获取文本框里的值，验证输入是否正确，执行相应的处理。具体代码如下：

```
string username = this.TextBox1.Text;   //获取输入的用户名
string password = this.TextBox2.Text;   //获取输入的密码
if ("admin"==username && "123456"==password)
{
   Response.Redirect("Success.aspx");  //重定向到登录成功页面。
} else {
   this.Label1.Text = "用户名或密码输入错误！";
}
```

好了，这个页面就处理完成了。很简单吧！

刚才我们提到还得有一个登录成功页面。下面就来创建一个登录成功的提示页面，系统验证通过以后，会跳转到这个页面。

在“项目资源管理器”中，我们右击第一行，即该项目的名称“FirstWebProject”，在弹出的快捷菜单中选择“添加”→“新建项”命令，弹出“添加新项”对话框，如图 1-26 所示。在“模板”区我们选择“Web 窗体”，在“名称”文本框中输入“Success.aspx”。然后单击“添加”按钮。

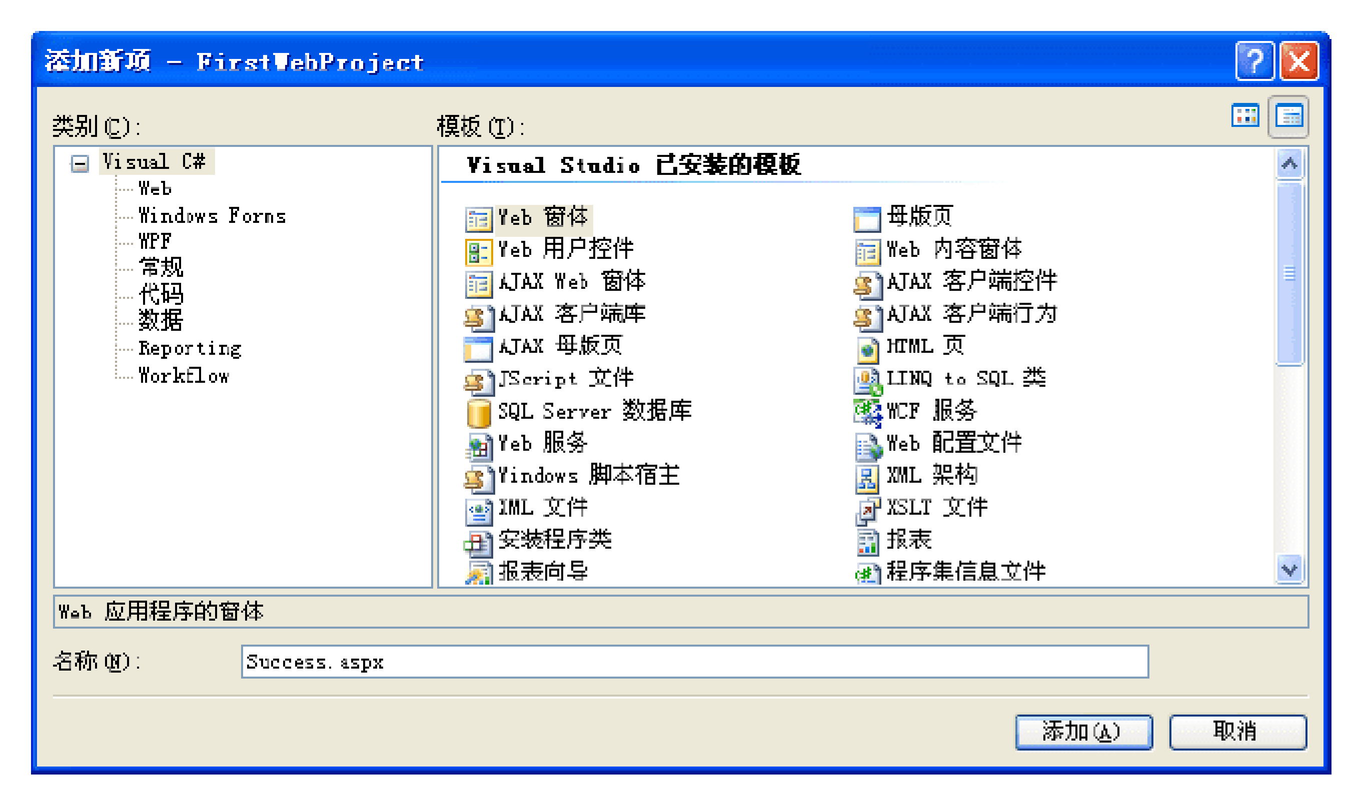


图 1-26 “添加新项”对话框

会自动打开 Success.aspx 文件，我们只在页面中的 body 部分添加如下一行代码即可：

```
<h2>恭喜你登录成功。</h2>
```

保存并关闭所有文档以后，整个程序就算开发完成。

我们来运行一下看看结果。

1.4.4 运行结果

按下键盘上的 F5 键，或者单击窗口工具栏里的“启动调试”按钮，会弹出“未启用调试”对话框，这里我们直接单击“确定”按钮启用调试。系统会自动启动浏览器，打开默认的登录页面。

在“用户名”文本框中随便输入一串字符，比如 abcd，在“密码”文本框中也随便输入一串字符，然后单击“登录”按钮，这样页面上就会提示“用户名或密码输入错误！”，如图 1-27 所示。

然后输入正确的信息试试，用户名 admin，密码 123456，单击“登录”按钮。好了，页面跳转到了 Success 页面，如图 1-28 所示。

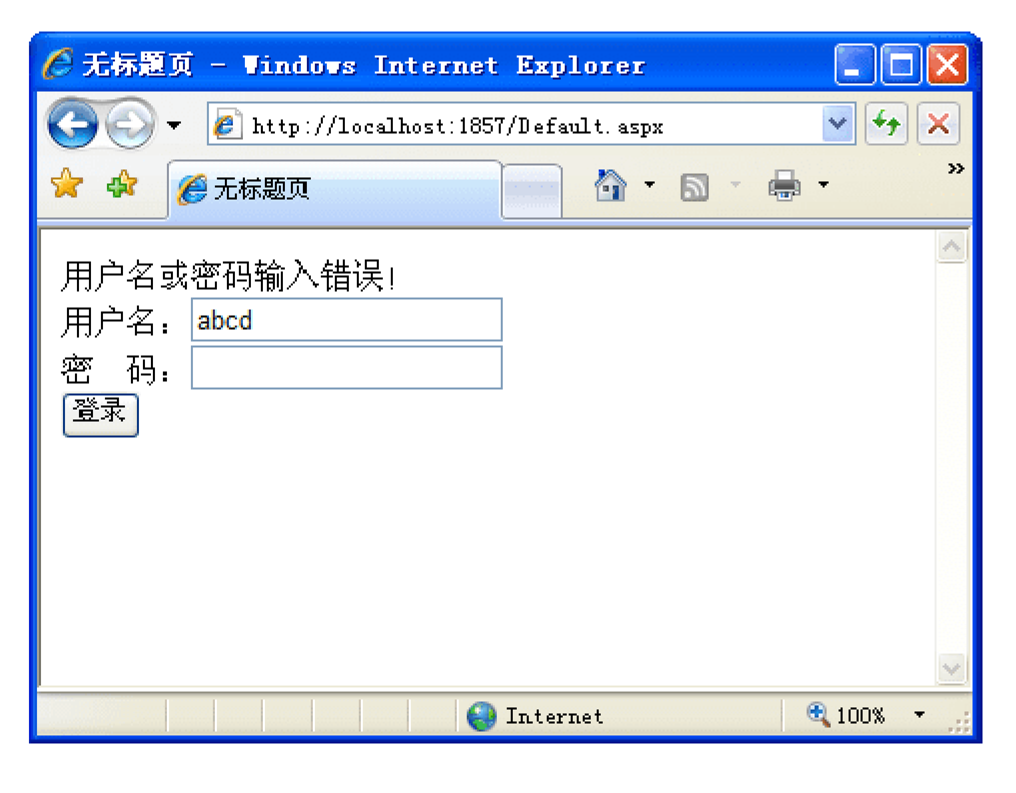


图 1-27　登录失败提示

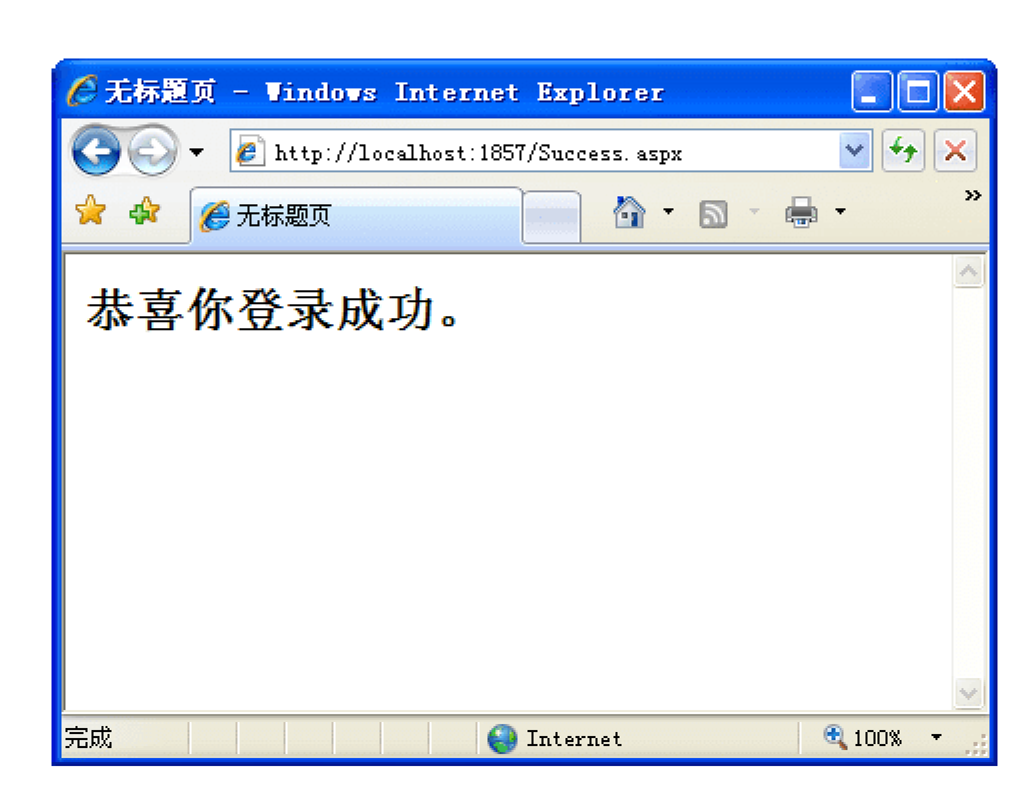


图 1-28　登录成功页面

1.4.5　实例分析



源码解析：

在实例中我们使用了 4 个服务器端控件：两个文本框，一个命令按钮和一个文本标签。双击命令按钮时，系统会自动为该控件创建一个单击事件处理程序。当用户激发该控件的单击事件时，系统自动执行该方法。在这个方法里，我们直接使用文本框的 Text 属性获取文本框里的内容。进行处理后，把处理结果直接赋给页面 Label 控件的 Text 属性，展示在页面上。

1.5　新建ASP.NET网站

上一节我们开发了一个简单的 ASP.NET 网站。那只算是开发了一个网站程序，还没有发布。就像我们要开公司——员工招来了，办公桌椅有了，资金什么的都到位了，但还需要找地方办公啊。所以我们还需要用服务器来存放网站程序。

当然，我们的办公地点还得有个唯一的地址，以免别人找我们公司时却跑到了别人的公司，白给别人拉了客户。



视频教学：光盘/videos/1/ConfigIIS.avi　　　　长度：3 分钟

1.5.1 基础知识——在IIS上创建新站点

通常，访问一个网站时都是直接输入域名的。但是使用 IP 地址照样能够访问一个网站。只不过一个服务器上只开一个网站有点浪费，所以还可以用 IP 根据不同的端口访问不同的网站。但是“IP + 端口号”的方式既啰嗦，又不容易记忆，所以出现了用域名定位网站的方式。

在 Windows Server 2003 + IIS 6 的 Web 服务器上，可以支持多个网站同时运行。IIS 6 同样是以“IP + 端口号 + 域名”三种方式组合来定位一个 Web 站点的。只要域名不同，所有的网站都可以同时使用一个“IP + 端口号”。

我们安装 IIS 6 的时候，系统会自动创建一个名为“默认网站”的站点。这个站点 IP 是“未分配”的，即服务器上所有 IP 都能访问，端口号为 80(HTTP 服务默认端口号)，域名为空。

1.5.2 实例描述

网站既然是 Web 应用程序，当然就要运行在 Web 服务器上。

在本章开始的时候，我们学习过安装和配置服务器环境，这里就要在这个服务器上给我们的网站安个“家”。

1.5.3 实例应用

【例 1-5】新建 ASP.NET 网站。

执行下面的步骤即可创建一个 Web 站点。

(1) 打开“控制面板”里的“管理工具”，双击运行“Internet 信息服务(IIS)管理器”，如图 1-29 所示。

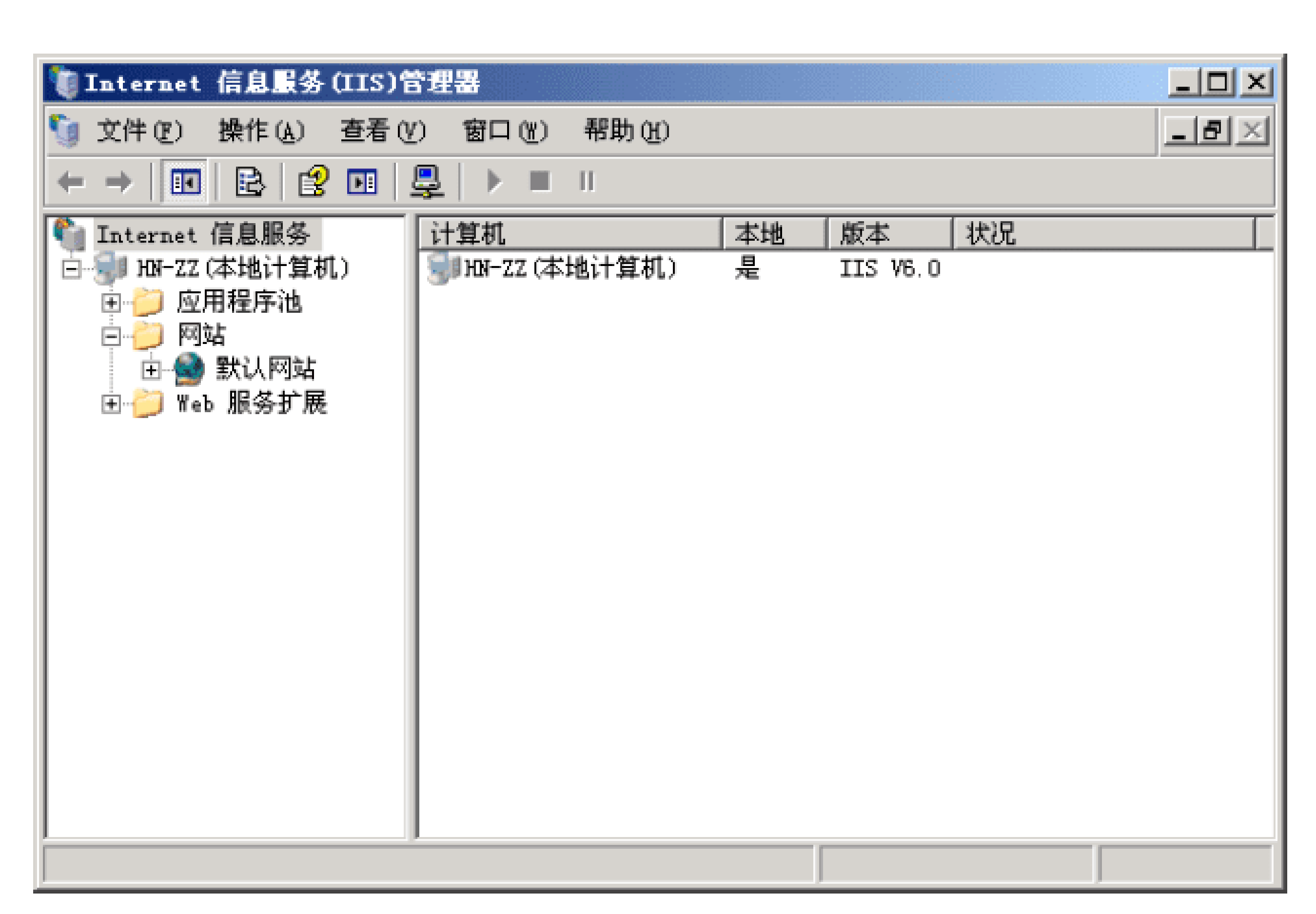


图 1-29 Internet信息服务(IIS)管理器

(2) 右击左侧栏中的“网站”节点，在弹出的快捷菜单中选择“新建”→“网站”命令，打开“网站创建向导”。单击“下一步”按钮，进入“网站描述”界面，如图 1-30 所示。

(3)　网站描述是指对网站的说明文字，即在 IIS 左侧栏中显示的网站节点名称。这里我们可以随便起名，并不影响网站的使用和运行。这里起名为 FirstWebSite。单击“下一步”按钮，进入“IP 地址和端口设置”界面，如图 1-31 所示。

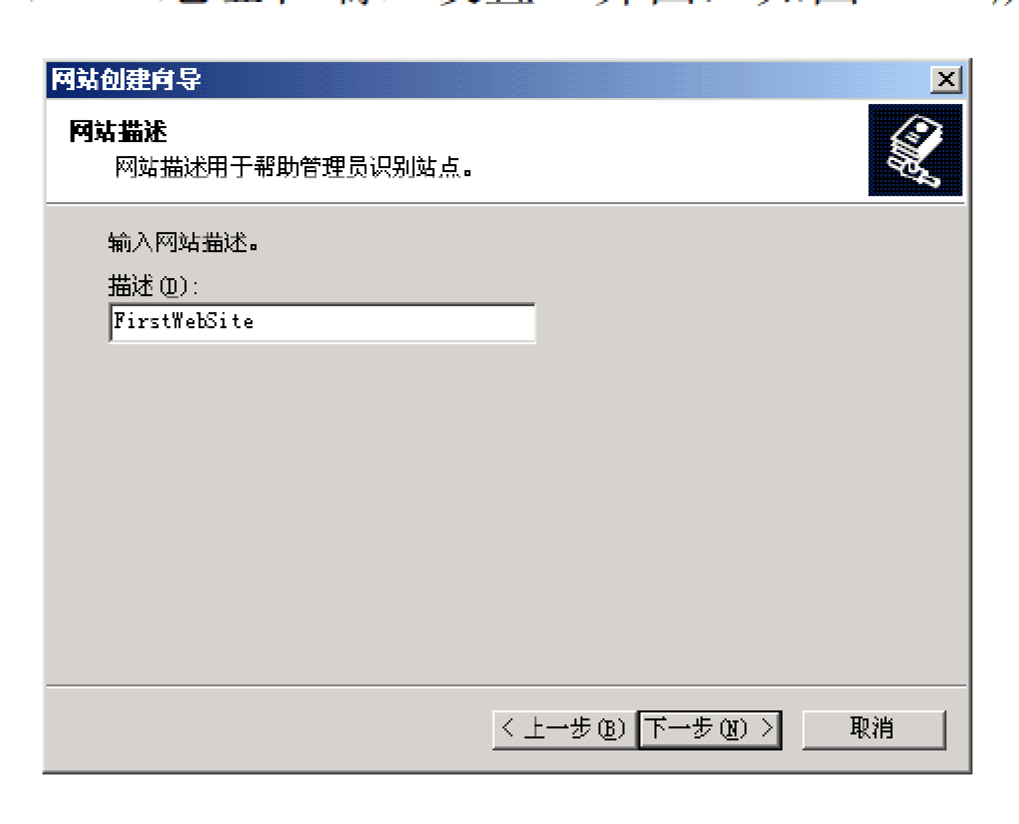


图 1-30　网站描述

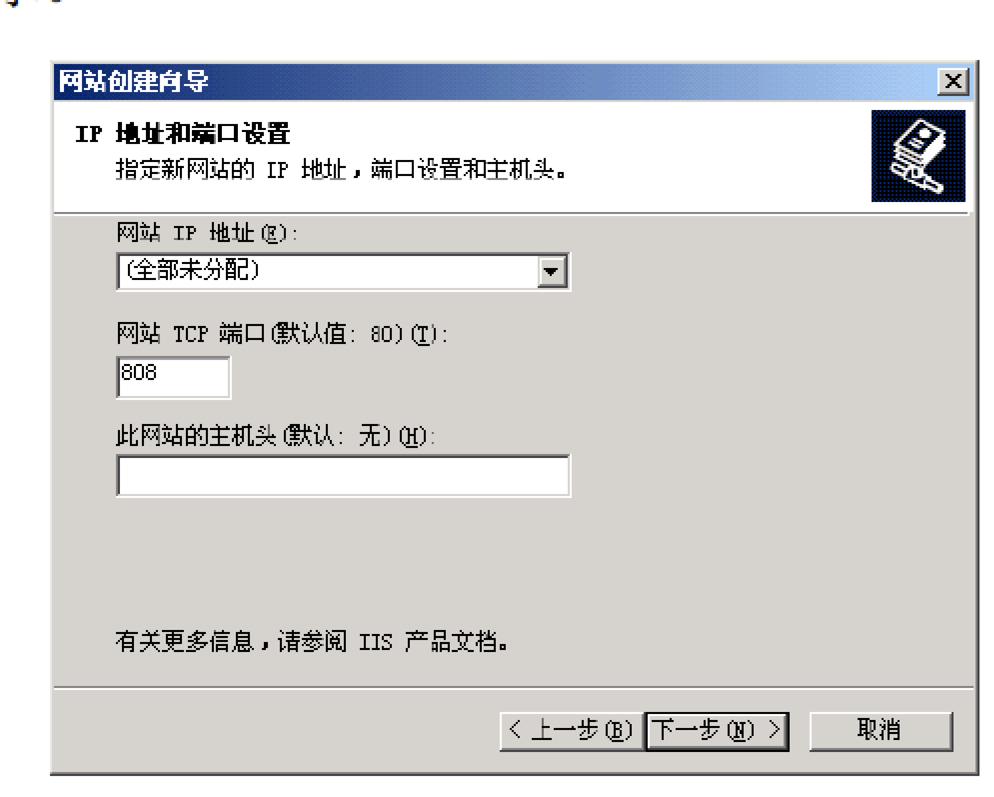
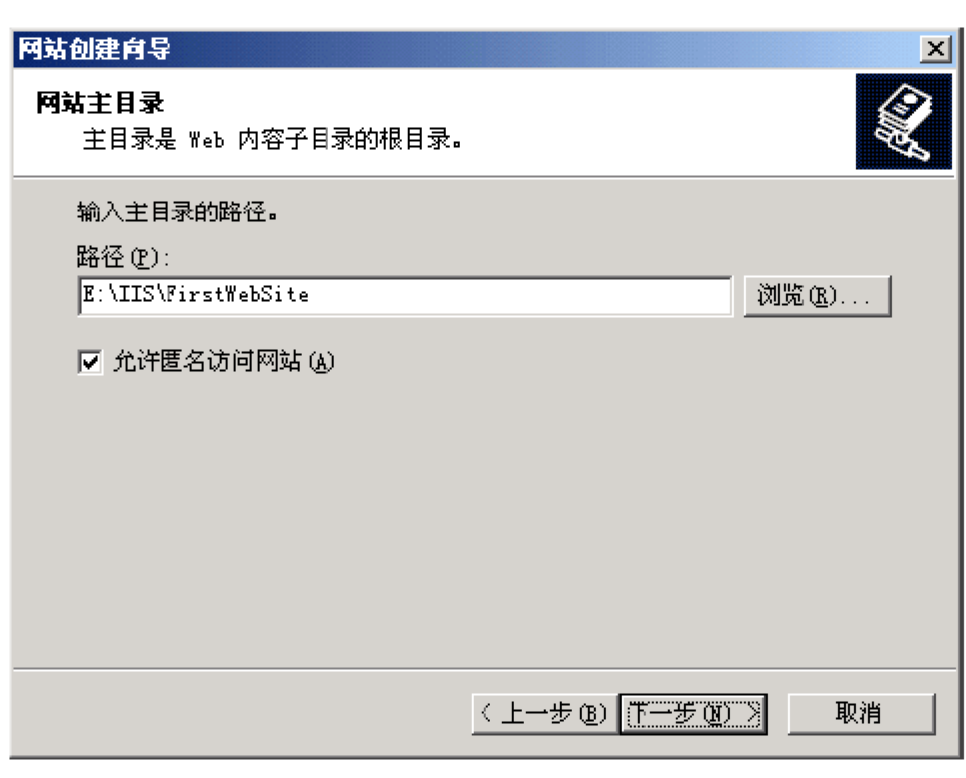
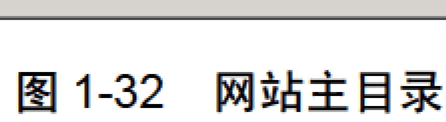


图 1-31　IP地址和端口设置

(4)　在这里，网站 IP 地址可以选择当前服务器上的网站所有的 IP 地址，也可以选择“全部未配置”，这样当服务器 IP 更换时就不用再次对这个站点进行设置了。因为我们是内网测试服务器，不能使用域名，所以用“IP + 端口号”的方式区分网站。前面说过 80 端口已经被“默认网站”占有，我们只能使用其他的端口。这里使用 808 端口来避免冲突。设置完后单击“下一步”按钮，进入“网站主目录”设置界面，如图 1-32 所示。

(5)　网站主目录是指网站程序文件所在的目录，这里可以单击“浏览”按钮，来选择服务器硬盘上的目录。选择完以后，单击“下一步”按钮，进入“网站访问权限”设置界面，如图 1-33 所示。




图 1-32　网站主目录

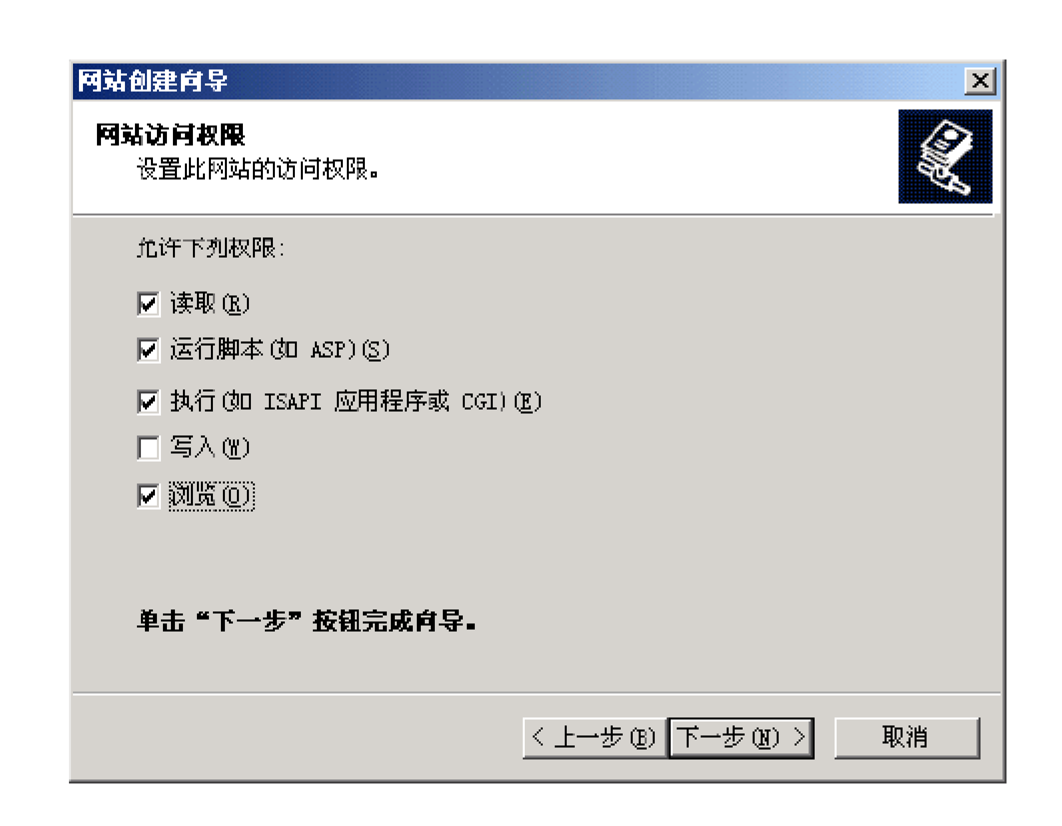


图 1-33　网站访问权限

(6)　关于网站访问权限，这里暂时就不深究相关的配置安全性了，选择除了“写入”以外的所有项，单击“下一步”按钮。出现“已成功完成网站创建向导”界面，如图 1-34 所示。

到这里，就说明我们的网站创建成功了。



图 1-34　已成功完成网站创建向导

1.5.4　运行结果

网站既然创建成功了，我们来测试一下。

我们在服务器的网站主目录里创建一个名为 index.html 的文本文件。用记事本打开它，写入如下一行代码：

```
<h2>Welcome!</h2>
```

然后在本地用浏览器访问该 Web 文件。访问路径为：

```
http://服务器 IP:808/index.html
```

例如访问服务器的 URL 为：

```
http://192.168.0.7:808/index.html
```

访问返回的结果如图 1-35 所示，看到了一行 2 号标题的欢迎语句，说明网站创建成功。

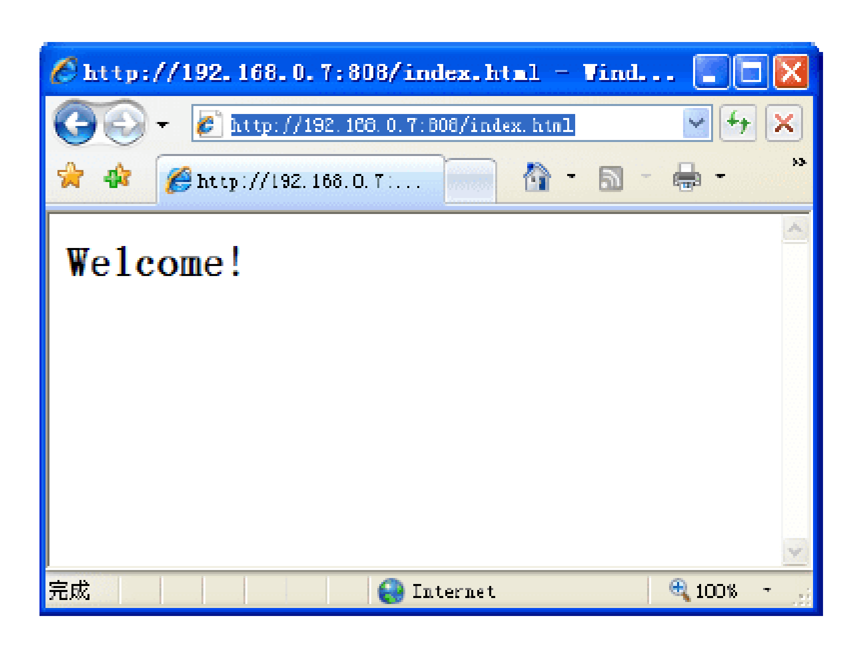


图 1-35　新建网站访问测试结果

1.6　发布ASP.NET网站

在前面的章节中，我们已经开发过一个简单的登录演示程序。虽然从功能来说它不能算是什么系统，但从结构上说，它已经是一个完整的 Web 程序了，能够发布和运行。

虽然我们已经调试运行过那个 Web 程序，但它仍然是我们创建的一个项目，并不是我们想要的纯粹的 Web 程序。因为它还能看到源代码，还能打开做修改。

作为一个软件厂商，想必不会主动把源代码交给客户吧。

客户需要的仅仅是一个 Web 程序而已。所以还要将我们的项目发布成一个纯粹的 Web 程序，并上传到服务器，完成发布、运行。


视频教学：光盘/videos/1/ConfigVS.avi　　　　**长度：4 分钟**

1.6.1　实例描述

在前面我们已经创建过一个简单的用户登录功能。虽然简单，但也还算是一个结构完整的 Web 程序。而后又新建并配置了一个 ASP.NET 站点，给 ASP.NET 应用程序找了一个“家”。

这一节就来发布我们的 Web 程序，并上传到服务器上。通过设置让我们的 ASP.NET 应用程序住进这个服务器之“家”。

1.6.2　实例应用

【例 1-6】发布 ASP.NET 网站。

首先，来发布我们的应用程序。

(1) 打开前面创建的 Web 应用程序 FirstWebProject，右击项目名称 FirstWebProject，在弹出的快捷菜单中选择“发布”命令，弹出“发布 Web”对话框，如图 1-36 所示。

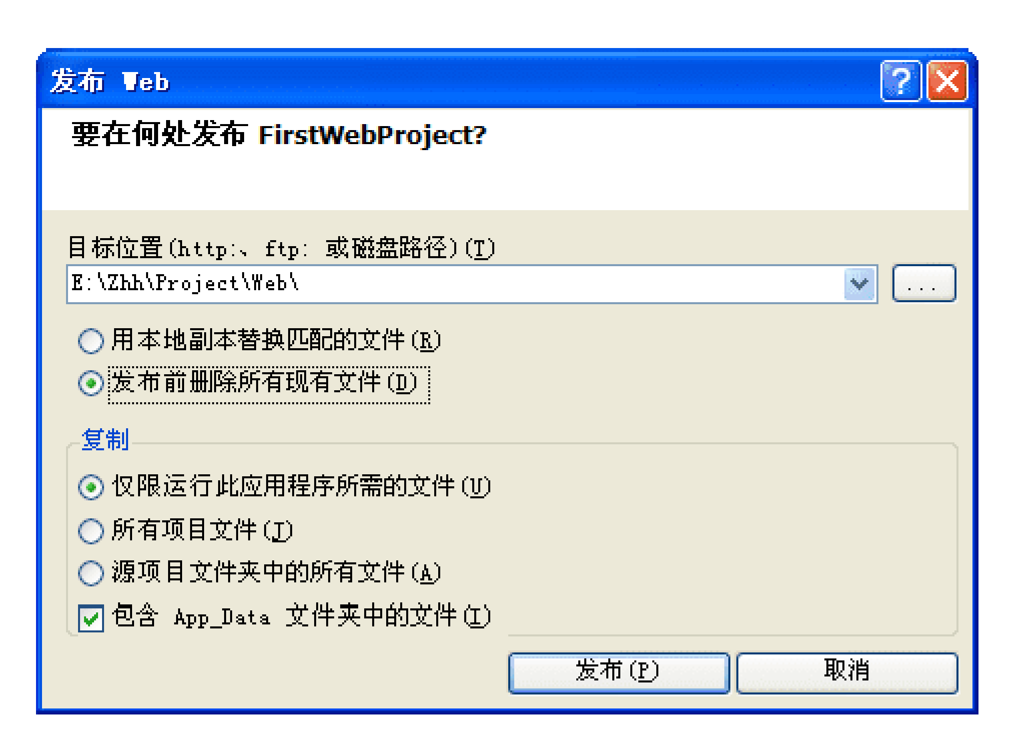


图 1-36　发布Web

(2) 在这个对话框中，选择我们要发布 Web 程序的目标路径，这里指向一个空的目录即可。在路径选择框下面的两项是说假如目标目录不为空做如何处理，这里选择删除目录下所有文件。“复制”选项组中是选择要发布的项目，这里我们选中“仅限运行此应用程序所需的文件”单选按钮，其他不必要的文件一概不要。当然，同时还要选中“包含 App_Data 文件夹中的文件”复选框，说不定里面会有一些数据文件，选中了会同时发布。设置完以后，单击“发布”按钮，“发布 Web”对话框将会消失，VS2008 的“输出”窗口显示出与发布有关的详细信息，如图 1-37 所示。

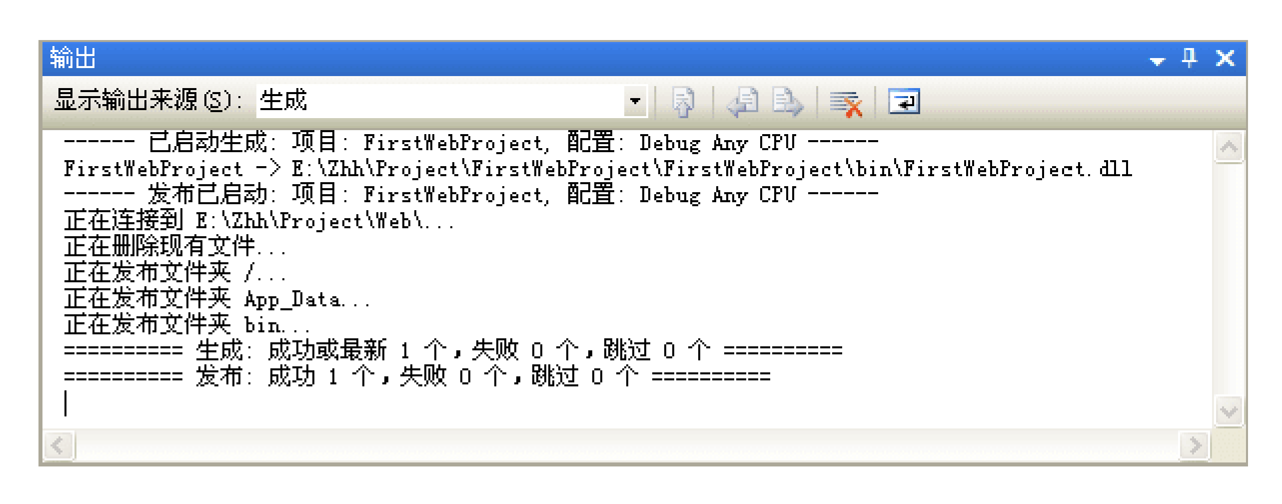


图 1-37　“输出”窗口

经过这些简单的操作之后，应用程序就发布成功了。我们来打开程序发布目标目录，可以看到结构非常简单，如图 1-38 所示。

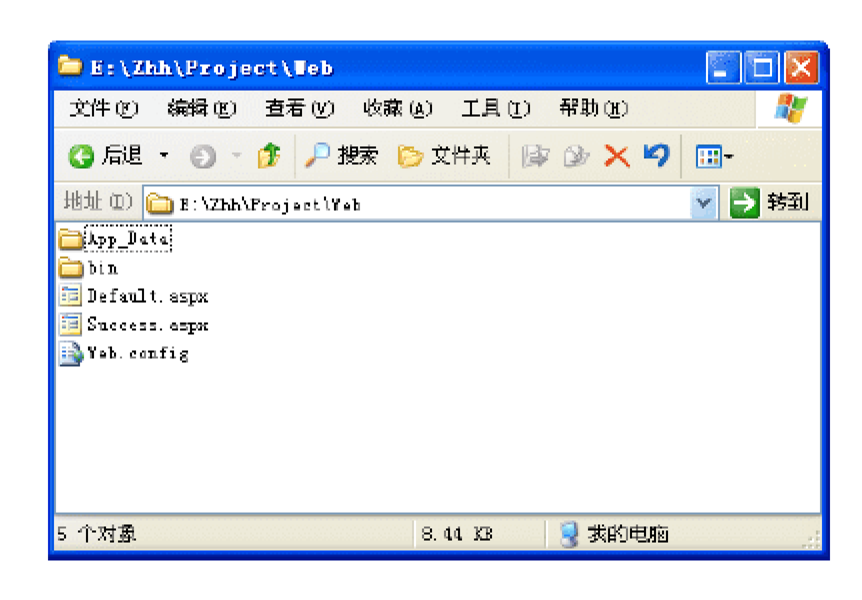


图 1-38　Web应用程序目录

- App_Data：数据文件目录，这里我们没有用到任何数据文件，所以是个空目录。
- bin：应用程序库文件目录，Web 应用程序引用的第三方库文件和应用程序编译成的库文件都在这个目录下面。
- 两个.aspx 文件：这就是网页文件了，供用户访问用的。
- Web.config：站点配置文件。

多么整洁的目录！

当然，因为这个程序简单，所以这里内容较少。一般应用程序还需要有存放图片、CSS 文件、JavaScript 文件的目录。

发布完了，我们还得上传到服务器上。一般 Web 服务器都是使用 FTP 方式上传，这里就不讲解 FTP 怎样配置和上传了。

所谓“上传”，就是把我们刚刚发布的那个目录里的东西全部复制到我们所创建的那个服务器 Web 站点 FirstWebSite 的主目录所指向的文件夹中，如图 1-39 所示。

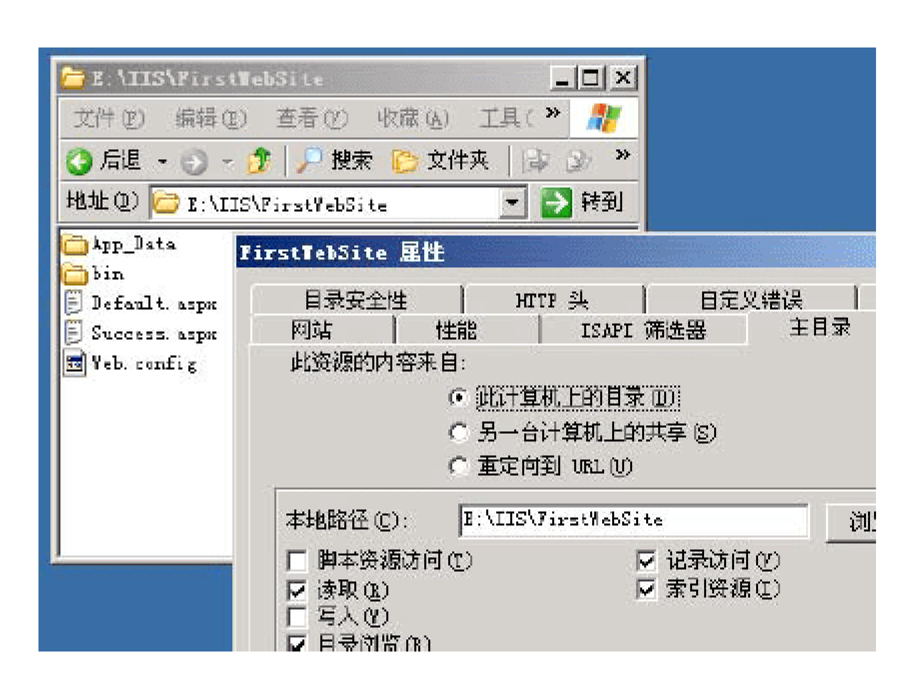


图 1-39　Web站点主目录

你可以使用任何办法。作者用的是局域网，所以直接共享了，复制过去就搞定。

1.6.3　运行结果

在本地打开浏览器，访问服务器中我们创建的网站。这里是 http://192.168.0.7:808/。访问结果如图 1-40 所示。

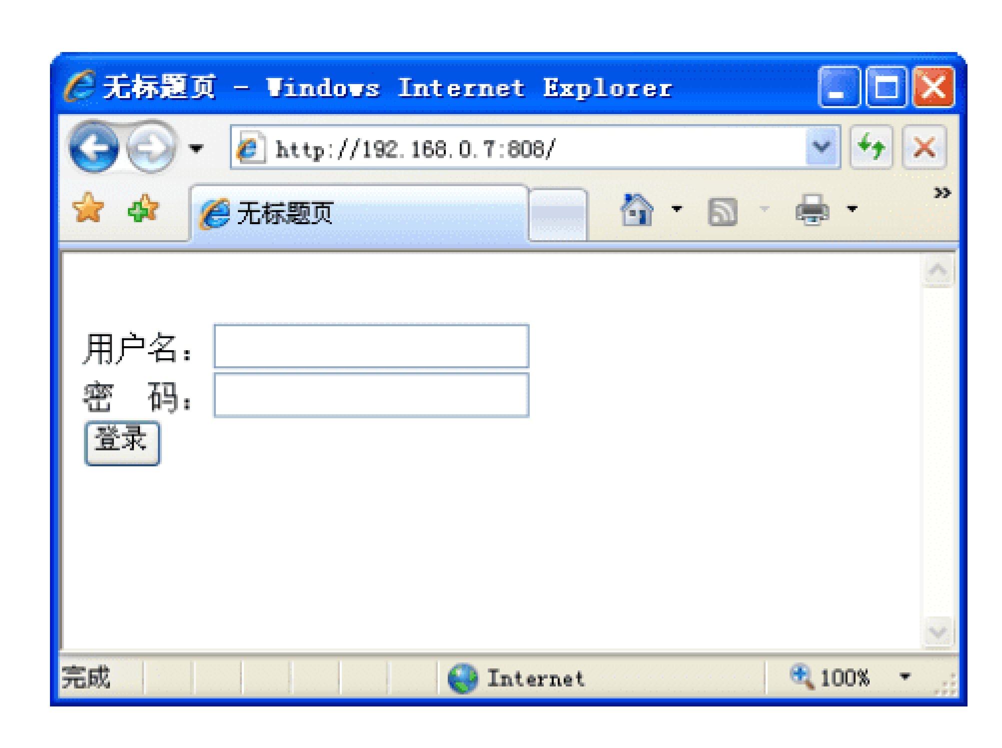


图 1-40　访问ASP.NET应用程序

效果是不是和我们在开发工具里测试的一模一样啊？

至此，我们的 ASP.NET 应用程序就发布成功了。

学会了上述内容，今后就可以开发功能更加完整的应用程序，发布到 Internet 上了。

所以读者现在应该会很有成就感了吧！

1.6.4　实例分析



源码解析：

上述实例首先把我们的 Web 项目发布到一个本地目录下，再将发布好的站点文件上传到 Web 服务器上指定站点的主目录下，这样就完成了网站的发布。

1.7　为ASP.NET网站分配应用程序池

任何应用程序都不可能是绝对完美的。没有任何一个人敢肯定他开发的程序没有一点 Bug。所以说程序在运行的过程中总会出现这样或那样的一些异常或错误，很可能就会导致程序死掉。

然而 Web 应用程序都是运行在 Web 服务器 IIS 上的，Web 应用程序的崩溃很可能会导致 IIS 服务器的故障，继而影响到 Web 服务器 IIS 上的其他应用程序。

IIS 作为一个优秀的 Web 服务器程序，它提供了多站点管理功能，当然对于这个问题它也给出了一套很恰到好处的解决方案，这就是“应用程序池”。

应用程序池能够很好地隔离 Web 应用程序，避免了应用程序之间的相互影响。


视频教学：光盘/videos/1/ConfigIIS.avi　　长度：3 分钟

1.7.1　基础知识——应用程序池

应用程序池实际上就是进程，一个应用程序池就是一个运行在内存里的进程。Web 站点可以设置挂载在哪个应用程序池上运行。

当然进程与进程间是互不影响的。所以一个 Web 应用程序崩溃不会影响到其他应用程序池里的 Web 应用程序，应用程序的故障只可能对它所在的应用程序池内的其他应用程序产生影响。这样就对整个 Web 应用程序进行了很好的隔离，增加了服务器的安全性和稳定性。

1.7.2　实例描述

一个 Web 服务器里可能运行多个不同的 Web 应用程序，我们不能保证别的站点都运行得十分正常。为了保证我们站点的安全性和稳定性，需要在 Web 服务器上单独为我们的站点建立一个应用程序池。

1.7.3　实例应用

【例 1-7】为 ASP.NET 网站分配应用程序池。

下面仍然来操作我们的服务器系统。

(1) 运行 IIS，在左侧列表里有一个与“网站”同级别的节点，名为“应用程序池”，展开后，下面就是 Web 服务器管理的所有应用程序池。IIS 初始只有一个默认的应用程序池 DefaultAppPool，如图 1-41 所示。

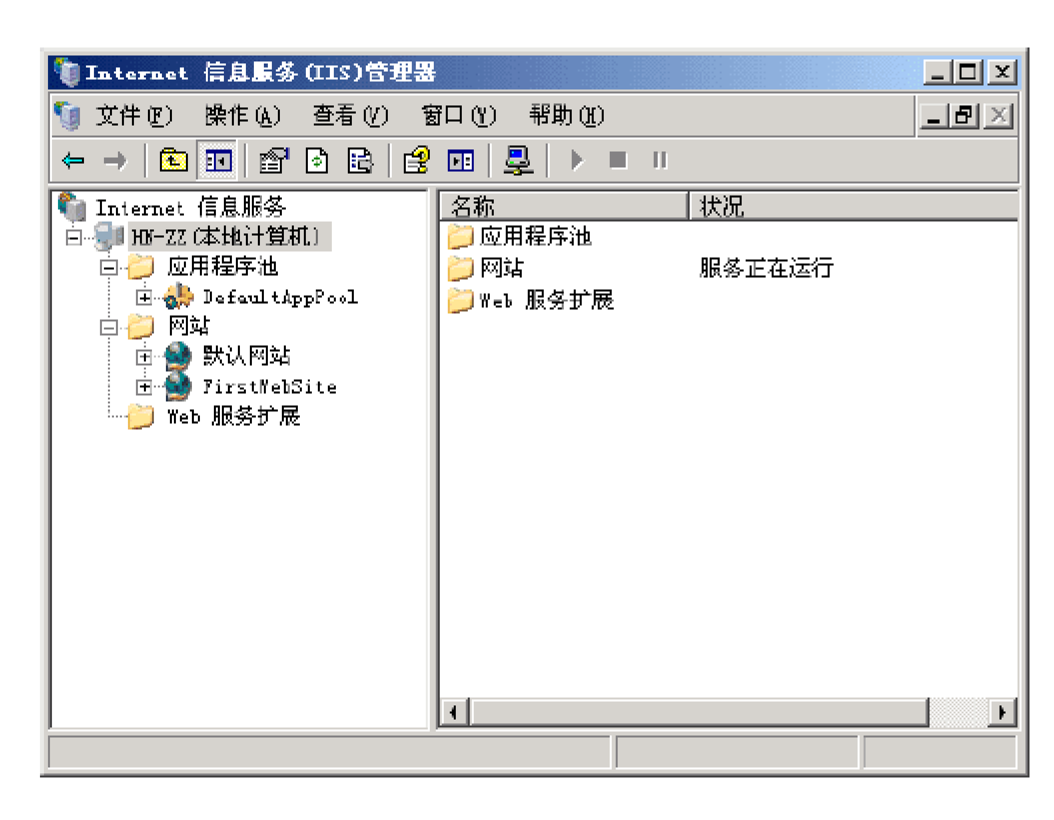


图 1-41　Internet信息服务(IIS)管理器

(2) 此时，我们来创建一个新的应用程序池。右击“应用程序池”节点，在弹出的快捷菜单中单击“新建”，在子菜单中单击“应用程序池”，弹出“添加新应用程序池”对话框，如图 1-42 所示。

图 1-42　“添加新应用程序池”对话框

(3)　在“应用程序池 ID”文本框中，为我们创建的应用程序池起一个名字 MySiteAppPool，下面的“应用程序设置”使用默认设置。然后单击“确定”按钮。在 IIS 左侧窗口内的“应用程序池”节点下就会多出一个我们刚才创建的应用程序池。

(4)　应用程序池创建完成后，我们将我们的站点挂载到这个应用程序池内，也就是设置站点在这个应用程序池内执行。右击我们的 Web 站点 FirstWebSite，在弹出的快捷菜单中选择“属性”命令，弹出“FirstWebSite 属性”对话框。

(5)　在“FirstWebSite 属性”对话框中单击展开“主目录”选项卡，如图 1-43 所示。

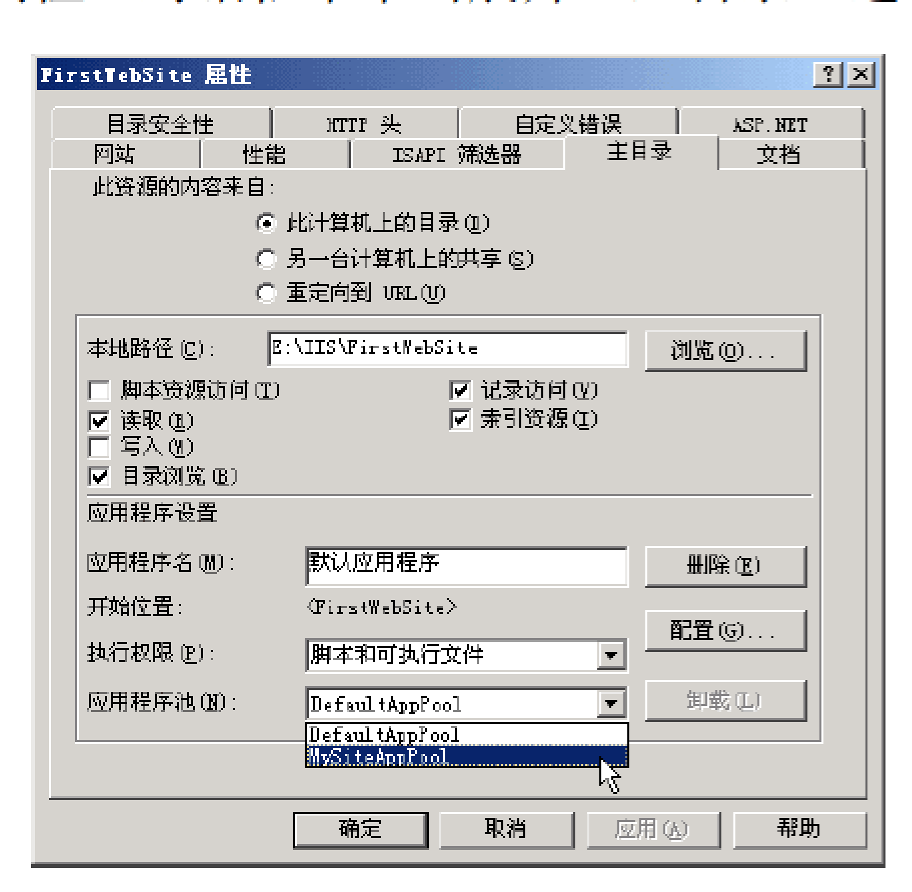


图 1-43　“主目录”选项卡

(6)　在下面的“应用程序池”下拉列表框中选择刚才我们创建的应用程序池 MySiteAppPool，然后单击“确定”按钮。

至此应用程序池就创建并设置成功了。

1.7.4　运行结果

对上述操作的结果需要验证一下，否则我们怎么知道是不是真的创建成功了呢？

前面说到“一个应用程序池就是一个运行在内存里的进程”，那我们就来监视一下任务管理器中的进程列表。

我们刚才只修改了 FirstWebSite 站点的应用程序池，所以这两个站点分别属于两个不同的应用程序池，所以它们运行的时候应该会有两个应用程序池进程出现在进程列表中。

下面我们来分别访问一下这两个站点。然后再来看任务管理器中的进程列表。果然出现了两个应用程序池进程，如图 1-44 所示。

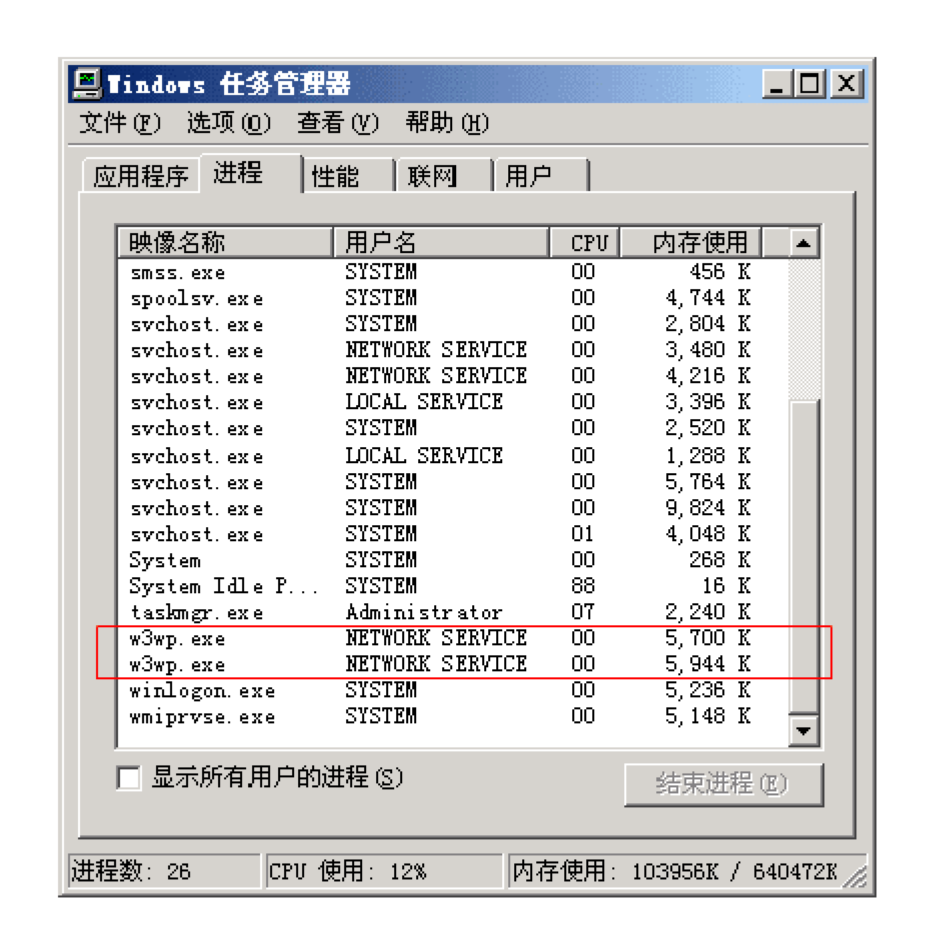


图 1-44 服务器进程管理器

1.7.5 实例分析


源码解析：

该实例比较简单，先创建一个应用程序池，再设置站点使用该应用程序池就行了。

应用程序池进程(w3wp.exe)默认情况下在一段时间内若处于空闲状态会自动关闭，以节省服务器资源。所以我们在某些时候打开进程管理器会看到少于应用程序池个数的 w3wp.exe 进程。当系统检查到有浏览器访问时，会自动启动相应的进程，来运行 Web 站点。

1.8 配置ASP.NET网站的访问权限

假如有一天，我们的网站发布了，在 Internet 上每天可能会有成千上万人来访问，当然，白花花的银子也滚滚而来地跑到我们面前。

然而突然有一天，网站崩溃了，数据全没了，许久以来耗费的心血在一夜间全部付诸东流。您会是什么心情呢？

检查一下是什么原因吧。

原来一个小学生下课没事打开电脑随便耍了几招，就把网站给“黑”了。

保证你听到这个消息后已经气死了，或者是后悔死了。

注意这只是“假如”而已。但如果对网站不认真设置访问权限，真正的麻烦就会找上门的。

还好我们的网站还没有发布，为了不至于将来被气死，先把网站设置好吧。可能我们还做不到绝对的无懈可击，但至少得先防止小学生的骚扰吧。


视频教学：光盘/videos/1/ConfigIIS.avi　　长度：3 分钟

1.8.1　基础知识——IIS站点权限

在 Internet 上，服务器其实也是一台电脑。只要开放足够多的权限，网络上的其他用户就可以像操作自己的电脑一样操作我们的服务器，结果可想而知。

如果我们的访客都是谦谦君子，当然就可以高枕无忧了。但访客真的都是谦谦君子吗？呵呵，不会的！所以需要进行安全配置。

IIS 给我们提供了一些基本的 Web 服务器安全配置选项。

- 脚本资源访问：选中该项时可以让用户访问程序源代码，即页面脚本。几乎没有人愿意把源代码暴露给访客，所以一般不要选择该项。
- 读取：允许用户访问站点的文件、目录及相关属性。这应该是必须选中的项，如果取消，任何资源将不能被访问。
- 写入：允许用户将文件直接写入服务器的目录中。如果开启此项，在最早的支持 HTTP 1.1 协议标准的 PUT 功能的浏览器中，可以直接用浏览器把文件上传到 Web 服务器上。当然，这样你的网站就不安全了。
- 目录浏览：开启该项，客户就能像在本地目录一样在浏览器查看网站目录了，他能看到站点里的所有文件，当然 App_Data 目录里的内容有专门设置的除外。
- 记录访问：选中该项，可以在日志里记录所有用户的访问信息，以备后来查看。可根据实际需求决定是否选择。
- 索引资源：使用系统提供的 Microsoft Indexing Service(微软索引服务)来对整个站点建立索引，一般小企业网站没有必要，这里不用选择。
- 执行权限：这是一个很重要的设置选项。它有三个子选项，“无”代表只能访问静态的文件，“纯脚本”是指只可以运行脚本代码，不可以运行可执行程序，当然“脚本和可执行文件”就什么都可以了。

1.8.2　实例描述

我们的站点现在还只是一个学习测试站点，所以只需要它运行脚本程序，而且暂时不需要记录日志。

所以这里就以最高的安全规则配置它，具体配置如下：脚本资源访问，当然不要；读取，是必需的，选中；写入，光听名字就不会要它；目录浏览，我们不想被人一眼望穿，去掉；记录访问，去掉；索引资源，不要；执行权限，我们只要“纯脚本”就行了。

1.8.3　实例应用

【例 1-8】配置 ASP.NET 网站的访问权限。

(1)　运行我们的服务器。

(2)　打开 IIS。

(3)　打开站点 FirstWebSite 的属性对话框，切换到“主目录”选项卡。

具体配置如图 1-45 所示。

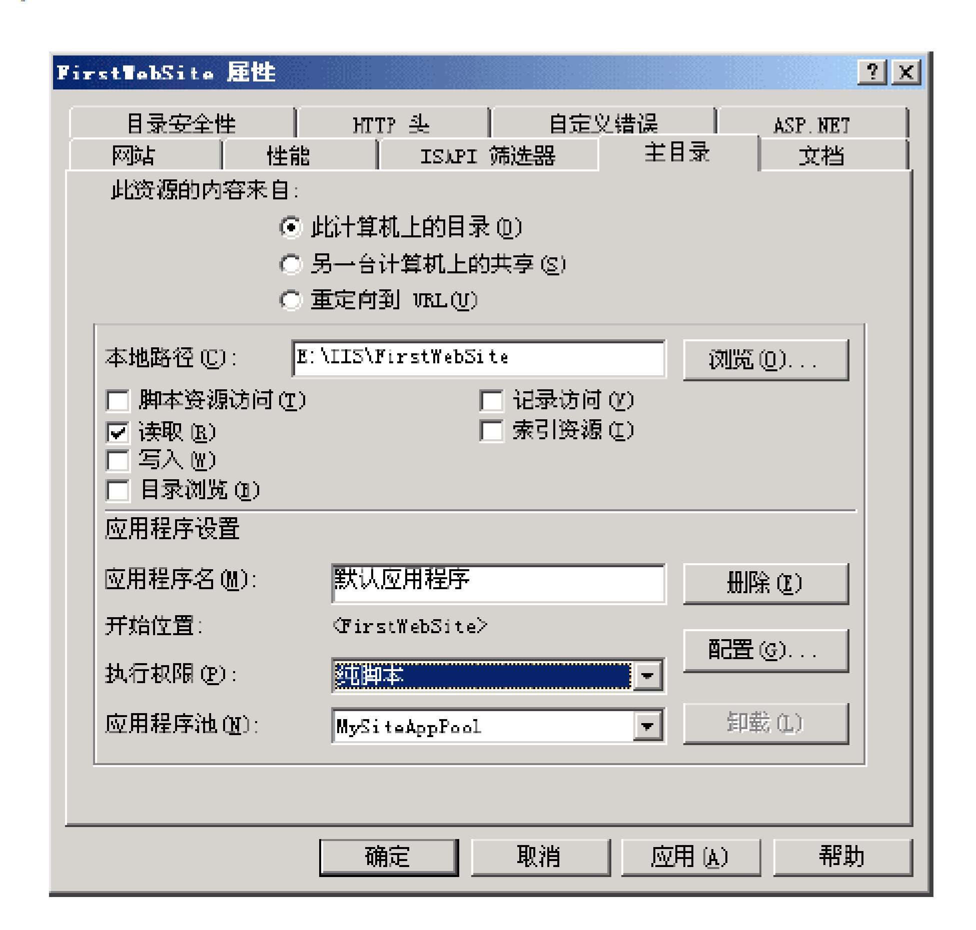


图 1-45 “主目录”选项卡

配置完以后单击“确定”按钮即可。

1.8.4 实例分析



源码解析：

本节说的是站点的访问权限设置，对于整个 Web 服务器安全性来说只是个皮毛。同样在属性对话框里我们还看到一个“目录安全性”选项卡，是配置站点用户的。也就是说任何 Web 应用程序都需要有对服务器系统的一定的操作权限，该权限就在这里设置。这有点扩大到服务器安全的配置了，我们暂且不再研究了。

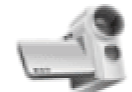
1.9 限定网站的带宽

前面说过，一个 Web 服务器可以运行很多网站。可能有 BBS、社区系统、视频系统、门户网站、资源下载站等。

本来一个服务器是有限的带宽、有限的连接数资源。但有人会开一个在线视频，没等来几个用户，带宽就被这一个站给占完了。或者有个很火的 BBS，在线人数成百上千，连接数满了，别的网站也打不开了。

前面我们讲过，应用程序池隔离了 Web 应用程序。确实不假，但是它是隔离应用程序的故障而已，对于公共资源的占用，它就无能为力了。

当然，IIS 对该问题也提供了一个很好的解决方案，这就是网站的性能设置。



视频教学：光盘/videos/1/ConfigIIS.avi　　**长度：**3 分钟

1.9.1　基础知识——了解网站带宽

运行在 Internet 上的主机(比如我们的服务器)，在与 Internet 接入的时候肯定有一定带宽限制。现在一些小服务器一般都是 3M，好点的是 10M，大型的数据机房就另当别论了。

不过带宽再高，也有上限。假如全世界几十亿台主机在同一时刻来访问你的服务器，我敢保证你的服务器百分之百地将会挂掉。

假如我们的服务器带宽是 10M，我们运行了几个网站。其中有一个客户挂的是一个下载站点，有很多资源。某一天有 50 个人一起上这个站点下载 VS2008 的安装程序，则平均每个人 200K 的带宽。基本还行！如果这样稳定的话，5 个小时左右就下载完了。

感觉怎么样，很爽吗？5 个小时啊，所有人都不能再访问这个服务器上的任何其他站点了，听清楚了，是任何其他站点。

这太不公平了吧！没办法。

假如因为这个网站太占资源，给撤了；然后又来了一个很火的 BBS，真的很强大，同时在线数经常在 500 个以上，而我们可怜的服务器因为性能原因允许同时在线人数 800。不过还行，勉强能运行。

但假如突然有一天，有一篇爆炸性新闻的贴子出现在这个 BBS 上，一度有数千用户争相点击。不用说，其他站点又没法使用了。

当然，为公平起见，我们会协商解决——让那个下载站最多占 3M 带宽，让这个 BBS 最多有 300 个连接。好了，现在即便同时有两个下载站和两个 BBS，我们的服务器还是有余力为其他 Web 站点提供服务。

这当然是个不错的解决方案！

1.9.2　实例描述

假设我们要开发一个地区性的门户系统，预计同时在线 300 人左右。

因为是门户，什么资源都有，所以可能还提供一些资源供下载，预计的资源多是些文档，大小在 10MB 以下。300 个在线的人可能会有 10%的人在下载东西，也就是 30 个人同时下载。

一般人下载 10MB 的东西最希望在三五分钟以内下载完成，所以我们预计平均让每个下载用户享有 50kB/s 左右的下载速度，这样 30 个人就是 1.5M 了。再有其他模块的访问带宽，预计我们的站点需要 3M 的带宽。

所以，我们的站点需要 300 个连接和 3M 的带宽。当然，既然是自己的服务器，我们想怎么设置就可以怎么设置了。

1.9.3　实例应用

【例 1-9】限定网站的带宽。

还是运行我们的服务器。

打开 IIS。

打开我们的站点 FirstWebSite 的属性对话框，选择“性能”选项卡。

选中“限制网站可以使用的网络带宽”复选框，在下面的微调框里手动输入 3072。注意单位是“千字节/秒”，也就是“kB/s”。3072kB/s 就是 3M 了。

然后对于下面的“网站连接”我们选中“连接限制为”单选按钮，在右边的微调框中输入 300。就是说最多 300 个连接，如图 1-46 所示。

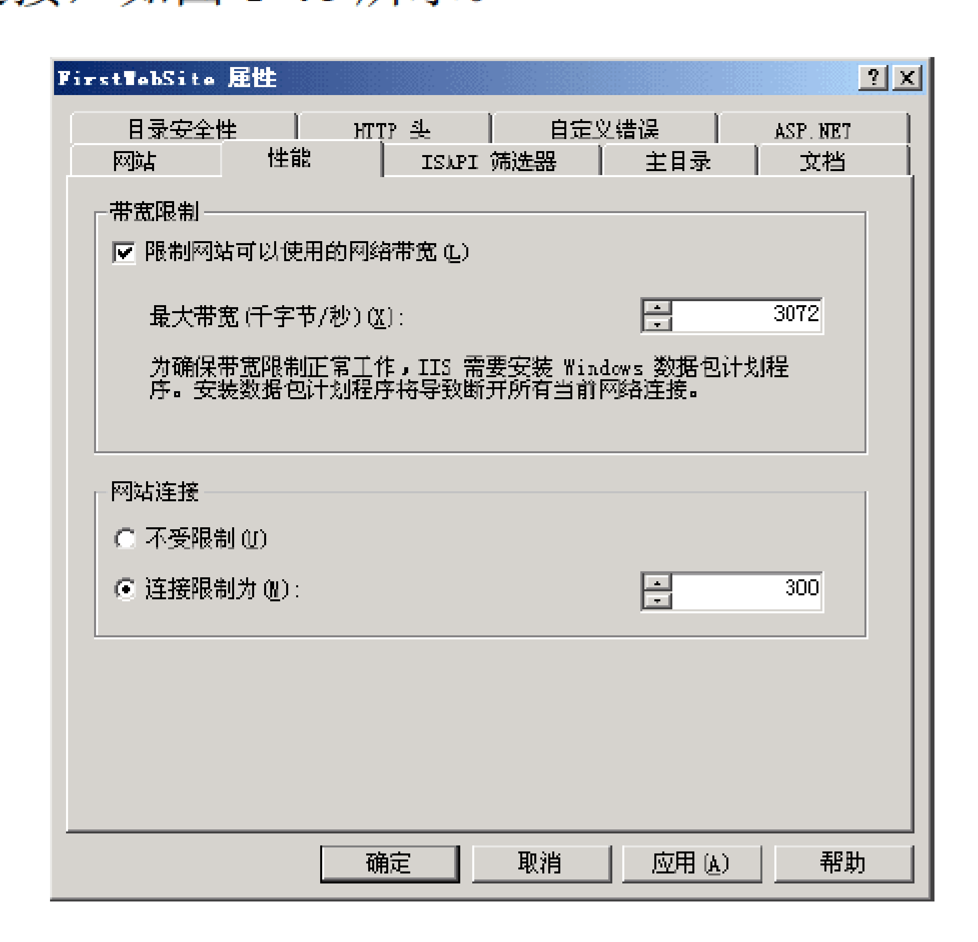


图 1-46 “性能”选项卡

这样就设置完成，以后就可以开发我们的网站了，相信它已经能经受住暴风雨的考验了。

1.9.4 实例分析



源码解析：

这个性能设置项是灵活的，具体的设置还要看具体的情况。所有的站点设置的数值之和可以大于服务器的性能上限。比如服务器的带宽是 10M，你可以设置 5 个网站都是 3M，但是不要设置单独一个站点都大于或等于 10M，那样就没意义了。这里我们作为开发人员，先了解一下，不然若我们把站点放到租来的空间上，同时两三个人打开就出现拒绝服务的情况，而怀疑是我们的程序问题，就太孤陋寡闻了。

1.10 常见问题解答

1.10.1 配置ASP.NET的环境仅装IIS和VS2008 行不行



配置 ASP.NET 的环境仅装 IIS 和 VS2008 行不行?

网络课堂：http://bbs.itzcn.com/thread-3560-1-1.html

请问配置 ASP.NET 的环境只安装 IIS 和 VS2008 行不行？还有个问题，.NET Framework

Web 程序员成功之路

SDK 各版本之间是不是只能安装一个？安装了 VS2008 还用安装.NET Framework SDK 吗？

【解决办法】：配置 ASP.NET 环境需要安装 IIS 和.NET Framework，但微软开发的 Visual Studio 2008 是一款强大的工具，它包含了.NET Framework SDK 3.5，所以安装 IIS 和 VS2008 就可以了。

.NET Framework 各版本之间是不冲突的，可以安装多个版本的.NET Framework，但建议安装最新版本的就可以了。

1.10.2　先安装VS2008，再安装IIS的补救办法



先安装 VS2008，再安装 IIS 的补救办法。

网络课堂：http://bbs.itzcn.com/thread-9850-1-1.html

我现在已经安装完 VS2008，但没有安装 IIS，有什么补救的办法呢？

【解决办法】：VS2008 安装过程中有个在 IIS 注册 Framework 的动作，如果先安装 VS2008，就必须重新注册，执行如下命令(注意参数)即可：

```
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\aspnet_regiis.exe -i
```

ASP.NET IIS 注册工具(Aspnet_regiis.exe)允许管理员或安装程序很容易地更新 ASP.NET 应用程序的脚本映射，以便指向与工具相关的 ASP.NET ISAPI 版本。此工具还可以用于显示所有已安装的 ASP 版本的状态、注册与工具配对的 ASP.NET 版本、创建客户端脚本目录，并执行其他配置操作。

1.11　习　　题

一、填空题

(1) 编译 C#程序所使用的工具是__________。

(2) Web 站点默认的端口号是__________。

(3) VS2008 创建的 C#代码文件默认扩展名是__________。

(4) ASP.NET 的默认主页文档名称是__________。

二、选择题

(1) 在 Visual Studio 2008 中，默认集成的.NET Framework 版本是__________。

A. 2.0　　B. 3.5　　C. 4.0　　D. 4.5

(2) C#代码文件编译过后的文件扩展名是__________。(多选)

A. .exe　　B. .cs　　C. .dll　　D. .jar

(3) ASP.NET 的 Web 项目生成以后的程序库文件存放在__________目录中。

A. bin　　B. App_Data　　C. dll　　D. 引用

(4) Visual Studio 2008 默认支持的语言有__________。(多选)

A. Visual C#　　B. Visual C++　　C. Visual Basic　　D. Visual Java

(5) IIS 6 中，应用程序池其实就是个应用程序进程。Windows Server 2003 系统里的 IIS 6 版本的应用程序池在任务管理器里的进程名称是__________。

A. iis.exe　　B. iis6.exe　　C. system.exe　　D. w3wp.exe

三、上机练习

上机练习：配置和熟悉 IIS 服务器。

虽然更新版本的 Windows 服务器系统和与其配套的 Web 服务器都已经推出。但当前市场的主流仍然是以 Windows Server 2003 系统和 IIS 6 组合为基础的 Web 应用程序平台。所以熟练地使用和配置 IIS 6 还是很有必要的。

IIS 6 的界面如图 1-47 所示。

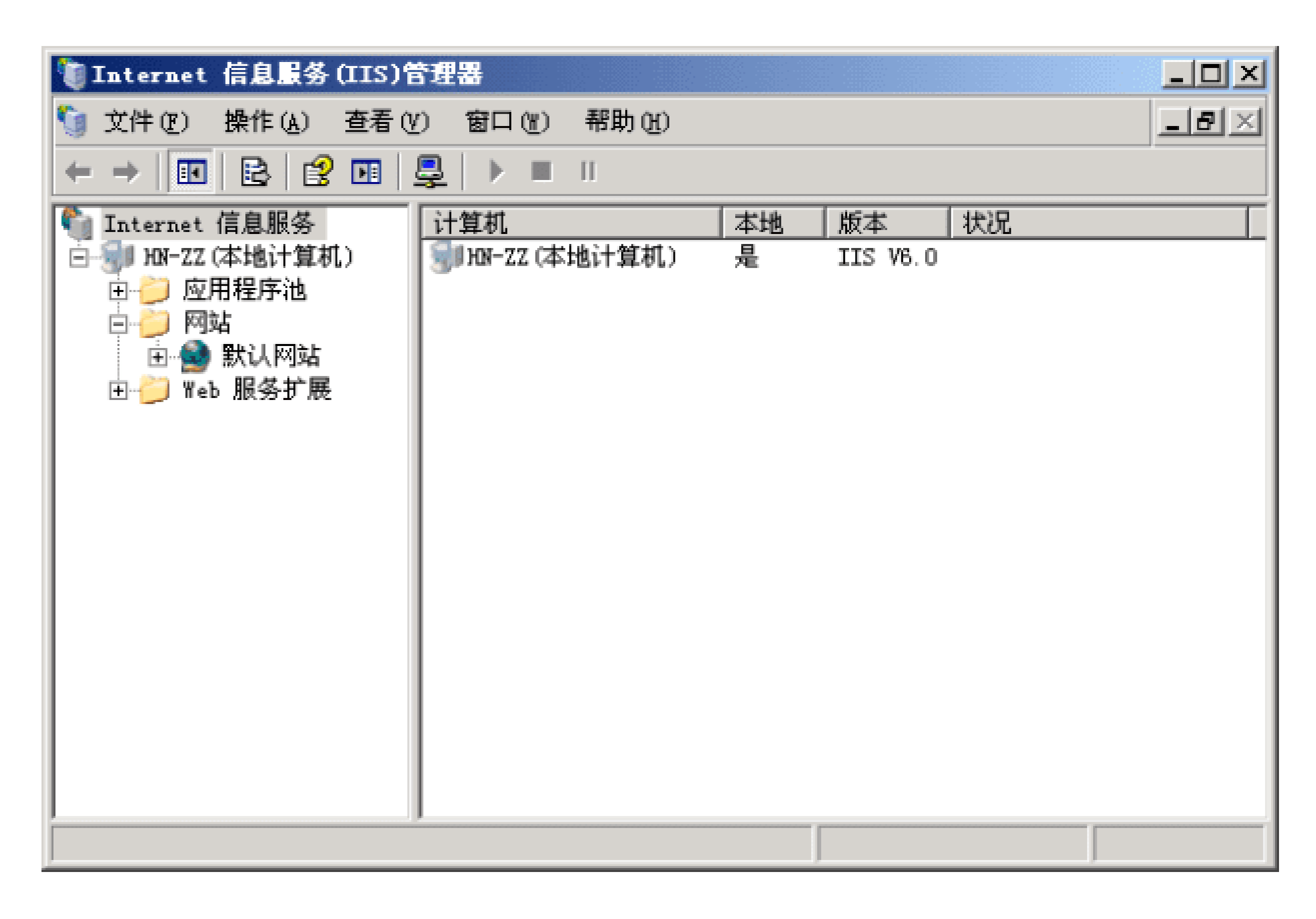


图 1-47　IIS 6



第 2 章　Web开发的必备技能

内容摘要：

Web 开发需要各种各样的技能，其中最基本的技能就是熟悉 HTML 表单和控件的使用。表单(Form)可以将用户(站点访问者)输入或者选择的信息提交到服务器上。

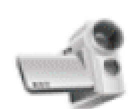
Web 开发除了基本的技能之外，还需要了解 Web 服务端的控件。像验证用户输入是否符合标准，除了可以用 JavaScript 脚本之外，还有更简单、更方便的方式，就是用验证控件。它只需要设置属性，就可以很快地在服务端验证是否符合标准。Web 服务端控件可以实现 HTML 控件所不能实现的功能。在本章中，我们将为大家讲解 Web 开发需要哪些技能。

学习目标：

- 了解并掌握 HTML 表单和控件的用法
- 掌握验证控件的用法
- 掌握 AdRotator 控件和 XML 文件的用法
- 掌握 Calendar 控件的用法
- 了解并掌握用户控件的编写过程及使用方法
- 了解并掌握导航控件以及站点地图的用法

2.1 模拟QQ的修改资料界面

最开始的网页都是静态网页，是基于 HTML 表单控件来进行开发的。像文本框、按钮、下拉列表之类的常用表单控件，都可以用来满足用户的基本需求。在本节中，我们就来了解一下 HTML 表单控件。

 视频教学：光盘/videos/2/HTML.avi　　长度：5 分钟

2.1.1 基础知识——HTML表单

表单标记是以<form>开头，以</form>结尾。语法如下：

```
<form method="get|post" action="URL"></form>
```

如上所示，<form>标记有两个属性：action 属性和 method 属性。

1. action属性

action 属性是<form>标记必需的，它用来指定表单提交时要采取的动作。用户填入表单的信息总是需要程序来进行处理，表单里的 action 就指明了处理表单信息的文件，这个文件一般是要对表单数据进行处理的相关程序地址，也可以是收集表单数据的 E-mail 地址，该 URL 所指向的服务器并不一定要与包含表单的网页是同一服务器，它可以是位于任何另外地方的一台服务器，只要给出绝对 URL 地址即可。示例代码如下：

```
<form id="form1" action="index.aspx"></form>
<form id="form2" action="http://www.itzcn.com/asp.net/yourname.aspx"></form>
```

2. method属性

method 属性有两个可选值，也就是说表单提供了两种数据传输方式——GET 和 POST。虽然它们都是数据的提交方式，但是在实际传输时却有很大的不同，并且可能会对数据产生严重的影响。为了使程序开发人员能更加方便地得到变量值，Web 容器已经屏蔽了二者的一些差异，但是了解二者的差异在以后的编程中会很有帮助。

技术文档	GET和POST的区别
<form>标记中的 GET 和 POST 方法，在数据传输过程中分别对应 HTTP 协议中的 GET 和 POST 方法。二者的主要区别如下。 (1) GET 用来从服务器上获得数据，而 POST 用来向服务器上传递数据。 (2) GET 将表单中的数据按照 variable=value 的形式添加到 action 所指向的 URL 后面，并且两者使用“?”连接，而各个变量之间使用“&”连接；POST 是将表单中的数据放在 form 的数据体中，按照变量和值相对应的方式，传递到 action 所指向的 URL。 (3) GET 是不安全的，因为在传输过程中，数据被放在请求的 URL 中，而现有的很多	

服务器、代理服务器或者用户代理都会将请求 URL 记录到日志文件中，然后放在某个地方，这样可能会有些隐私的信息被第三方看到。另外用户也可以在浏览器上直接看到提交的数据，一些系统内部消息将会一同显示在用户面前。而 POST 的所有操作对用户来说都是不可见的。

(4) GET 传输的数据量小，这主要是因为受 URL 长度的限制；而 POST 可以传输大量的数据，所以在上传文件时只能使用 POST。

2.1.2 基础知识——HTML表单控件

上面学习了如何创建一个 HTML 表单，本小节将介绍一些常用的 HTML 表单控件，它们通常被包含在表单中，从而实现各种交互功能。

1. 单行输入文本框

单行输入文本框主要用于提供少量的文本输入，输入的文本可以是任何字符或数字。示例代码如下：

```
<input type="text" name="yourname" />
```

2. 普通按钮

普通按钮的示例代码如下：

```
<input type="button" value="link"/>
```

3. 提交按钮

提交按钮用来将表单(Form)里的信息提交给表单中 action 所指向的文件。提交按钮是表单中不可缺少的控件，只有当用户用鼠标单击提交按钮后，浏览器才把表单数据发送给服务器。没有提交按钮，表单数据就不会被发送出去。一个表单中可以有多个提交按钮，但它们的名称应该互不相同。用户可以单击其中的任何一个，表单数据都将被提交。示例代码如下：

```
<input type="submit" value="sub" name="submit1" />
```

上述代码中，属性 value 的值是提交按钮上的提示字符串，默认值是“submit”或“提交查询内容”。提交表单时，提交按钮的“名称/值”对同样作为表单数据被传送到服务器中。

4. 密码输入框

在 Web 页面上输入密码也是通过一个单行文本框来实现的，所以从外观上看起来，与单行输入文本框没有区别，但与单行文本框不同的是，在输入密码时，所输入的字符并不像单行输入文本框那样按照原样显示出来，而是将所有的字符用“*”符号代替，即输入的字符是不可见的，从而起到保密作用。示例代码如下：

```
<input type="password" name="yourpw">
```

从上述代码中可以看出，输入密码框控件与输入文本框控件的创建上只有 type 属性值设置不同。

5. 多行输入框

可以输入多行文字。示例代码如下：

```
<textarea name="yours" cols="50" rows="3"></textarea>
```

6. 下拉框(select)

可以做单选，也可以做多选。示例代码如下：

```
<select name="food">
    <option value="rice">大米
    <option value="noodles">面条
    <option value="Apple">苹果
</select>
```

如果变成多行显示，示例代码如下：

```
<select name="food" multiple="multiple">
    <option value="rice">大米
    <option value="noodles">面条
    <option value="apple">苹果
</select>
```

7. 单选按钮

单选按钮在 Web 页面上显示为一个空心圆圈，当用户在某一圆圈上单击鼠标时，圆圈中会出现一个实心圆点，表示该项被选中。再次单击该圆圈时，圆点消失，表示取消选中该项。示例代码如下：

```
<input type="radio" name="food" value="food">食物<br>
```

8. 复选框

复选框在 Web 页面上显示为一个小方框，可以用鼠标单击小方框进行选择或者取消选择。被选择项的方框中会被打上一个勾，表示被选中。当一个复选框单独使用时，复选框就像一个开关，它的取值只有两种可能：开(On)或关(Off)。如果用户选中一个复选框，当表单被提交时，浏览器会以“变量名=On”的形式，将该复选框的“名称/值”对传递给服务器。如果用户没有选中该复选框，则浏览器在提交表单时将忽略该复选框。示例代码如下：

```
<input type="checkbox" name="check" id="check"/>
```

通常情况下，复选框不是单独使用，而是成组使用，这样就可以构成一个多项选择。复选框的主要作用是让用户从多个可选项目中可同时选择多个，这一点是与单选按钮不相同的。

9. 隐藏控件

隐藏控件不会在网页上显示出来，所以在网页上用户看不到任何此控件存在的迹象。这种控件不是用来让用户看的，而是为 Web 开发者提供的。隐藏控件是一个虽然不显示但其值会随着表单被提交的控件。Web 开发者通常使用这种控件类型存储浏览器和服务器之间交换的信息。示例代码如下：

```
<input type="hidden" name="op" value="end" />
```

这里应该注意的是，由于隐藏控件不会被显示，所以用户在访问时无法输入相应的值，它的值是由 Web 开发人员进行设置的。

10. 图像控件

可以使用 input 控件创建一个图像控件，单击该控件之后，表单立即被提交。所以可以通过简单的设置(属性 src 指向一个事先处理好的图片)将其作为提交按钮来使用。示例代码如下：

```
<input type="image" src="logo.gif" name="image"/>
```

也可以直接用 img 控件创建。示例代码如下：

```
<img src="logo.gif" name="image"/>
```

11. 文件控件

可以使用 input 控件创建文件控件，该控件带有一个文本框和一个浏览按钮，通常用于文件的上传。用户可以在文本框中输入文件路径，也可以单击浏览按钮，找到你要的文件。示例代码如下：

```
<input type="file" name="file1" value="上传文件" />
```

用户要使文件上传成功，必须注意以下几点：

- 文件控件必须出现在<form>标记内。
- 必须为文件控件指定 NAME 属性的值。
- <form>标记的 method 属性的值必须设置为“post”。
- <form>标记 ENCTYPE 属性的值必须设置为“multipart/form-data”。

12. 重置按钮

如果用户在填写好表单后或者填写了一部分表单内容后想清除自己所填写的内容，那么可以单击重置按钮。当用户单击重置按钮后，浏览器会将表单中已经输入的数据全部清空，所有控件都将恢复到所设置的初始值或默认值。示例代码如下：

```
<input type="reset" name="reset1" value="重置" />
```

重置按钮与提交按钮非常相似，属性 value 的值是重置按钮上的提示字符串，默认值是“reset”或“重置”。通常情况下，每一个表单应用提供一个提交按钮和一个重置按钮，并将它们排放在表单的最下方，以便于用户在填写完表单内容后进行提交。

2.1.3　实例描述

好久没有写过静态网页了，过去一直是在 Visual Studio 2008 环境下写程序，实现功能，偶尔会在页面中加些样式，看着不会太难看。

现在要做的这个模拟 QQ 的修改资料界面，主要用到的是 HTML 表单常见的几个元素，如文本框、下拉列表、单选按钮、复选框、提交按钮、重置按钮、多文本输入框(文本域)、密

码输入框等。接下来就一起设计这个界面吧。

2.1.4 实例应用

【例 2-1】模拟 QQ 的修改资料界面。

(1) 首先打开 Dreamweaver 工具，然后新建一个 HTML 网页，命名为“QQ.html”，接下来开始写代码。

(2) 创建一个 5 行 3 列的表格。将第一行第三列往下合并 3 个表格。将第四行和第五行往右合并两个单元格。

(3) 在第一行第一列输入“用户账号”文字，第一行第二列添加一个 Label 控件，给它一个默认值为“123456789”，在第三列的第一个单元格中插入图片“Snap8.gif”，将它作为 QQ 头像来显示，在它下面加个超链接“更改头像”。代码如下：

```
<tr>
   <td width="86">用户账号：</td>
   <td width="246"><label>123456789</label></td>
   <td width="243" rowspan="3" align="center">
      <div><img src="Snap8.gif" /></div><div><a href="#">更改头像</a></div>
   </td>
</tr>
```

(4) 在第二行第二列插入一个超链接“使用 Email 地址作为账号”，代码如下：

```
<tr>
   <td> </td>
   <td><a href="#">使用 Email 地址作为账号</a></td>
</tr>
```

(5) 在第三行第二列添加一个文本框，作为 QQ 用户的昵称。代码如下：

```
<tr>
   <td>用户昵称：</td>
   <td><input type="text" /></td>
</tr>
```

(6) 在第四行第二列添加一个文本域，作为 QQ 用户的个性签名。代码如下：

```
<tr>
   <td height="27">个性签名：</td>
   <td colspan="2">
   <textarea name="qianming" rows="3" id="qianming" style="width:400px;">
   </textarea>
   </td>
</tr>
```

(7) 在第五行第二列添加一个复选框，属性值为“始终显示个性签名”。代码如下：

```
<tr>
   <td> </td>
```

```
    <td colspan="2"><input type="checkbox" />始终显示个性签名</td>
</tr>
```

(8) 创建一个 1 行 4 列的表格，表格宽度与上面表格的宽度一致，并插入相应的图片。代码如下：

```
<table width="589" border="0">
<tr>
    <td width="82" height="30">会员阶段：</td>
    <td width="71"><img src="Snap5.gif" /></td>
    <td width="97" align="right">QQ 等级：</td>
    <td width="321"><img src="Snap4.gif" /></td>
</tr>
</table>
```

(9) 接着再创建一个 5 行 6 列的表格，合并单元格，里面重点用到了下拉列表。添加一个下拉列表，代码如下：

```
<select><option>男</option><option>女</option></select>
```

这个表格的整体代码如下：

```
<table width="589" border="0">
<tr>
    <td width="92" height="27">性 别：</td>
    <td width="61">
    <select><option>男</option><option>女</option></select>
    </td>
    <td width="48">年龄：</td>
    <td width="106"><input type="text" width="50px"/></td>
    <td width="60">省份：</td>
    <td width="196">
    <select><option>河南</option><option>湖北</option></select>
    </td>
</tr>
<tr>
    <td width="92">生 肖：</td>
    <td width="61"><select><option>马</option></select></td>
    <td width="48" >血型：</td>
    <td width="106"><input type="text" width="50px"/></td>
    <td width="60">生日：</td>
    <td width="196">
    <select><option>1</option></select>月
    <select><option>1</option></select>日
    </td>
</tr>
<tr>
    <td>毕业院校：</td>
    <td colspan="3"><input type="text" width="200px"/></td>
    <td>星座：</td>
    <td><select><option>水瓶座</option></select></td>
```

```
</tr>
<tr>
    <td>个人主页：</td>
    <td colspan="3"><input type="text" width="200px"/></td>
    <td>职业：</td>
    <td><select><option>材料包装</option></select></td>
</tr>
<tr>
    <td >个人说明：</td>
    <td colspan="5">
    <textarea name="qianming2" rows="3" id="qianming2" style=" width:400px;">
    </textarea>
    </td>
</tr>
</table>
```

(10) 最后加上一个提交按钮，代码如下：

```
<div style="margin-top:20px;">
    <input type="submit" value="提交" />
</div>
```

2.1.5 运行结果

按 F12 键运行代码。结果如图 2-1 所示。



图 2-1 模拟QQ的修改资料界面

2.1.6　实例分析



源码解析：

该实例主要讲解了几种常见的表单元素，如文本框、下拉列表、单选按钮、复选框、提交按钮、重置按钮、文本域和密码输入框等。用表格对它们进行布局，会使界面显得比较美观。

整个表格放在<div></div>里，设置了它的上间距(margin_top)和左间距(margin_left)，最好是显示在页面的中间位置。然后将“提交”按钮单独放在一个 div 中，设置其上间距和左间距。

2.2　非数据库的网站管理后台登录界面

任何一个较完善的系统都离不开后台管理部分，甚至一般的系统也都有后台管理系统，只有管理员先进行登录，才能享受一些特殊的权利，对系统进行维护。

在本节我们将制作一个与数据库无关的网站后台登录界面。



视频教学：光盘/videos/2/Button.avi　　　　**长度：**13 分钟

2.2.1　基础知识——网站登录

网站登录界面通常包含用户名文本输入框、密码输入框和登录按钮这三个控件。

2.2.2　实例描述

不管是小茅屋，还是一栋大楼，它们都有进入口。所谓进入口，就是大门了。每一个网站系统，不管是前台，还是后台，都有一个进入口，就是我们所说的登录界面。别小看这个登录界面，它可是有很多值得我们大家学习的地方，比如说登录权限、登录错误 3 次锁定账号等。那么，在本节中，我们来学习一下如何设计登录界面。

2.2.3　实例应用

【例 2-2】非数据库的网站管理后台登录界面。

(1)　打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 Login.aspx 页面标题为“网站管理后台登录界面”。

(2)　创建一个两行两列的表格，设置表格的 ID 为“Tb_login”。

(3)　在第一行第二列添加一个文本框，ID 为“Txt_LoginUser”，边框颜色 BorderColor 为“#FFFFCC”，边框样式 BorderStyle 为“Ridge”。在第二行第二列添加一个文本框，ID 为“Txt_password”，边框颜色 BorderColor 为“#FFFFCC”，边框样式 BorderStyle 为“Ridge”，

显示文本的属性 TextMode 为“password”。代码如下：

```
<table width="277" height="77" border="0" id="Tb_login">
<tr>
    <td width="84">用户名：</td>
    <td width="179">
    <asp:TextBox ID="Txt_LoginUser" runat="server" BorderColor="#FFFFCC"
      BorderStyle="Ridge"></asp:TextBox>
    </td>
</tr>
<tr>
    <td>密码：</td>
    <td>
    <asp:TextBox ID="Txt_password" runat="server" BorderColor="#FFFFCC"
      BorderStyle="Ridge" TextMode="Password"></asp:TextBox>
    </td>
</tr>
</table>
```

(4) 然后添加“登录”图片按钮和“关闭”图片按钮。代码如下：

```
<asp:ImageButton ID="ImageButton1" runat="server"
  ImageUrl="~/网站后台登录_files/login001.jpg" onclick="ImageButton1_Click" />
 
<asp:ImageButton ID="ImageButton2" runat="server"
  ImageUrl="~/网站后台登录_files/login002.jpg" />
```

(5) 在项目中添加一个类，命名为“Login”。然后判断用户是否登录成功。代码如下：

```
public static bool Islogin(string username,string pwd)
{
    if (username=="admin" && pwd=="admin")
    {
        return true;
    }
    return false;
}
```

(6) 当单击“登录”按钮时，提示是否登录成功。代码如下：

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    if (login.Islogin(Txt_LoginUser.Text, Txt_password.Text))
    {
        ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('登录成功!')</script>");
    }
    else
    {
        ClientScript.RegisterStartupScript(
          this.GetType(),"","<script>alert('登录失败!')</script>");
```

```
    }
}
```

2.2.4　运行结果

按 Ctrl+F5 键运行程序，效果如图 2-2 所示。



图 2-2　非数据库的网站管理后台登录界面

2.2.5　实例分析



源码解析：

本案例设计了一个网站的登录界面，包括一个“用户名”文本输入框、一个“密码”文本输入框、一个“登录”图片按钮和一个“关闭”图片按钮。

还创建了一个判断登录的 Login 类，它的静态方法 Islogin()完成验证功能。在该方法中仅仅提供了一种思路，实际应用时需要结合数据库。

2.3　制作带验证功能的注册表单

ASP.NET 提供了一组验证控件和一种易用且功能强大的检错技术。使用这些控件和技术，开发人员可以轻松检查用户输入信息的有效性。比如初始值验证、比较验证、范围验证、正则表达式验证等。通过本节的学习，读者将可以了解使用验证控件和各种验证技术来检查用户输入信息的有效性的方法。

 **视频教学：**光盘/videos/2/RequiredFieldValidator.avi　　　　长度：6 分钟

2.3.1 基础知识——验证控件

ASP.NET 提供了用于验证控件(如文本输入框)内容的验证控件。如果用户输入的内容不满足验证控件的条件，则验证控件将显示错误提示信息。ASP.NET 常用的 5 个验证控件如下。

- RequiredFieldValidator：验证控件的内容是否等于预先设置的初始值。
- CompareValidator：比较两个控件的内容，或控件的内容和指定的值是否满足指定的关系。
- RangeValidator：验证控件的内容是否在一个指定的范围内。
- RegularExpressionValidator：验证控件的内容是否匹配指定的正则表达式。
- CustomValidator：开发人员可以通过自定义的客户端函数或服务端事件来验证控件的内容。

这几个验证控件都有一个共同的属性 ControlToValidate，它必须指定一个被验证的控件，属性值为该验证控件的 ID。如果未指定该属性的值，则发生编译错误。

另外，ASP.NET 还提供了 ValidationSummary 控件，该控件用来处理其所在页面上的所有验证控件的错误消息。然而，ValidationSummary 控件不是验证控件。

ErrorMessage 和 Text 属性都可以指定验证控件的错误提示消息。Text 属性的优先级高于 ErrorMessage 属性。如果同时设置了这两个属性的值，则验证控件优先显示 Text 属性的值。

Display 属性指定控件错误消息的显示方式。它的值为 ValidatorDisplay 枚举的值。可以为以下 3 个值。

- None(不显示方式)：验证控件不显示错误消息，并且不占用网页上的空间。
- Static(静态显示方式)：始终占用网页上的空间，不管是否显示验证控件的错误消息。
- Dynamic(动态显示方式)：只有当显示验证控件的错误消息时，才占用网页上的空间。

如果要使验证控件的错误消息不显示在网页上，而是显示在 ValidationSummary 控件中，则需要把验证控件的 Display 属性值设置为“None”。

2.3.2 实例描述

假如你想开车去某个地方旅行，但是你不知道路，这时候就要用到 GPS 了，知道这是什么玩意儿吧——导航系统，能引导你成功到达目的地，很方便吧。

在 ASP.NET 中有一个类似于 GPS 的东西，就是验证控件。用户在注册的时候，可以按照正确的操作步骤填写信息。下面就来学习一下如何制作带验证功能的用户注册表单。

2.3.3 实例应用

【例 2-3】带验证功能的注册表单。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 Form_zhuce.aspx 页面标题为“带验证功能的注册表单”。

(2) 创建一个 7 行 2 列的表格。从第一行第一列往下六行依次插入文本“用户名”、“密

码”、“确认密码”、“年龄”、“邮箱”、“身份证号”。从第一行第二列往下六行依次插入文本框，分别命名为“username”、“pwd”、“surepwd”、“age”、“email”、“idCard”。

(3) 验证用户名不能为空。添加 RequiredFieldValidator 验证控件，设置 ErrorMessage 属性值为“用户名不能为空”，ControlToValidate 属性值为“username”，即为对应文本框的 ID。示例代码如下：

```
<tr>
    <td class="style1" align="center">用户名</td>
    <td width="180">
    <asp:TextBox ID="username" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
      runat="server" ErrorMessage="用户名不能为空！"
      ControlToValidate="username">
    </asp:RequiredFieldValidator>
    </td>
</tr>
```

(4) 验证密码不能为空。添加 RequiredFieldValidator 验证控件，设置 ErrorMessage 属性值为“密码不能为空”，ControlToValidate 的属性值为“pwd”，即为对应文本框的 ID。示例代码如下：

```
<tr>
    <td class="style1" align="center">密码：</td>
    <td>
    <asp:TextBox ID="pwd" runat="server"></asp:TextBox>
    <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
      runat="server" ErrorMessage="密码不能为空！" ControlToValidate="pwd">
    </asp:RequiredFieldValidator>
    </td>
</tr>
```

(5) 验证确认密码同密码一致。添加 CompareValidator 验证控件，设置 ErrorMessage 属性值为“密码前后输入不一致！”，ControlToValidate 的属性值为“surepwd”，ControlToCompare 属性值为“pwd”。示例代码如下：

```
<tr>
    <td class="style1" align="center">确认密码：</td>
    <td>
    <asp:TextBox ID="surepwd" runat="server"></asp:TextBox>
    <asp:CompareValidator ID="CompareValidator1" runat="server"
      ErrorMessage="密码前后输入，不一致！" ControlToCompare="pwd"
      ControlToValidate="surepwd">
    </asp:CompareValidator>
    </td>
</tr>
```

(6) 验证年龄是否超出范围。添加 RangeValidator 验证控件，设置 ErrorMessage 属性值为“年龄不在规定范围内”，ControlToValidate 的属性值为“age”，最大值 MaximumValue 为“100”，

最小值 MinimumValue 为“0”，类型 Type 为“Integer”。示例代码如下：

```
<tr>
    <td class="style1" align="center">年龄：</td>
    <td>
    <asp:TextBox ID="age" runat="server"></asp:TextBox>
    <asp:RangeValidator ID="RangeValidator1" runat="server"
      ErrorMessage="年龄不在规定范围内！" ControlToValidate="age"
      MaximumValue="100" MinimumValue="0" Type="Integer">
    </asp:RangeValidator>
    </td>
</tr>
```

(7) 验证邮箱是否有效。添加RegularExpressionValidator验证控件，设置ErrorMessage属性值为“邮箱格式不符！”，ControlToValidate的属性值为“email”，ValidationExpression属性值为“\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*”。示例代码如下：

```
<tr>
    <td class="style1" align="center">邮箱：</td>
    <td>
    <asp:TextBox ID="email" runat="server"></asp:TextBox>
    <asp:RegularExpressionValidator ID="RegularExpressionValidator1"
      runat="server" ErrorMessage="邮箱格式不符！" ControlToValidate="email"
      ValidationExpression="\w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*">
    </asp:RegularExpressionValidator>
    </td>
</tr>
```

(8) 验证身份证号是否有效。添加 RegularExpressionValidator 验证控件，设置 ErrorMessage 属性值为“不是有效的身份证号！”，ControlToValidate 的属性值为“idCard”，ValidationExpression 属性值为“\d{17}(\d|x)”。示例代码如下：

```
<tr>
    <td class="style1" align="center">身份证号：</td>
    <td>
    <asp:TextBox ID="idCard" runat="server"></asp:TextBox>
    <asp:RegularExpressionValidator ControlToValidate="idCard"
      ID="RegularExpressionValidator2" runat="server"
      ErrorMessage="不是有效的身份证号！" ValidationExpression="\d{17}(\d|x)">
    </asp:RegularExpressionValidator>
    </td>
</tr>
```

(9) 在最后一行分别添加“注册”和“重置”两个按钮。

2.3.4 运行结果

运行程序，效果如图 2-3 所示。

图 2-3　带验证功能的注册表单

2.3.5　实例分析



源码解析：

验证控件用起来简单、方便，可以很快地对用户输入的信息进行检测。该案例主要用到的验证控件有初始值验证控件、比较验证控件、范围验证控件、正则表达式验证控件，这几个验证控件也是比较常用的。

验证控件最关键的属性就是 ControlValidate，必须为它设置属性值，属性值就是指定被验证的控件的 ID；其次就是设置错误提示消息了，一般用 ErrorMessage。

2.4　实现页面随机效果显示

这里我们将看到一个例子，它使用控件在一个页面上显示广告，通过刷新页面，页面上的图片会自动地更换。



视频教学：光盘/videos/2/AdRotator.avi　　　　**长度：**5 分钟

2.4.1　基础知识——AdRotator控件和XML文件

1. AdRotator控件

AdRotator 控件用来显示一个广告图片序列。此控件使用一个 XML 文件来存储广告信息。下面我们来了解一下 AdRotator 控件的属性。

- AdvertisementFile：包含广告信息的 XML 文件。
- BorderColor：为广告设置边界颜色，如 BorderColor="#FFFFFF"将边界设为白色。
- BorderWidth：边界的宽度，以像素为单位，如 BorderWidth="1px"。
- KeywordFilter：对广告类别进行过滤。
- Target：单击广告时 URL 的目标窗口。如 Target="_new"，表示每当广告图像被单击都会弹出一个新的窗口。"_top"是默认值。

2. XML文件

XML 具有很高的灵活性，但是必须遵循一定的结构，才能保证 XML 文档具有规范性。XML 文档结构可以分为三个部分：序言、主体和尾声。其中，序言和主体是每个 XML 文档必须保留的，而尾声则可以根据用户的需要来使用。

(1) 序言

序言是 XML 文档的开始，用来表示对 XML 数据进行编译的开始及描述字符的编码方法，并为 XML 解析器和应用程序提供其他一些配置线索。

示例代码如下：

```
<?xml version="1.0" encoding="编码" standalone="yes/no"?>
<!--注释-->
```

(2) 主体

主体就是 XML 文档描述数据的地方，通常由标记、元素、属性等构件组成。

(3) 尾声

XML 文档的尾声部分内容包括注释、处理指令和/或紧跟元素树后的空白。由于大多数应用程序在文档根元素的结束标记处就结束了，而不再对尾声进行任何处理，所以尾声部分对于 XML 文档来说不起任何作用。用户可以根据需要选择是否添加尾声。

2.4.2 实例描述

通过使用 ASP.NET 携带的 AdRotator 服务器控件，ASP.NET 中的广告可以随时显示出来。AdRotator Web 服务器控件在一系列可单击的广告条之间循环，并且允许某些广告条优先于其他广告条。广告可以通过具有预定义架构的 XML 文件链接到该控件，也可以通过你自己创建的自定义逻辑链接到该控件。

2.4.3 实例应用

【例 2-4】实现页面随机效果显示。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 Random_Ads.aspx 页面标题为"实现页面随机效果显示"。

(2) 创建一个包含着广告细节的 XML 文件，来存储广告条图像位置、用于重定向的 URL 以及相关属性。通过使用 XML 文件格式，可以创建和维护广告清单，而不必在对某一广告进

行更改时改变应用程序的代码。可以使用 XML 设计器中的“AdRotator 计划文件”模板创建 XML 文件。

XML 文件的主要代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>

<Advertisements>
   <Ad>
      <ImageUrl>expotclogo.png</ImageUrl>
      <NavigateUrl>http://www.163.com/</NavigateUrl>
      <AlternateText>链接到网易主页</AlternateText>
      <Keyword>网易</Keyword>
      <Impressions>25</Impressions>
   </Ad>
   <Ad>
      <ImageUrl>logo_china_yasha.gif</ImageUrl>
      <NavigateUrl>http://www.163.com/</NavigateUrl>
      <AlternateText>链接到网易主页</AlternateText>
      <Keyword>网易</Keyword>
      <Impressions>25</Impressions>
   </Ad>
   <Ad>
      <ImageUrl>logo_v1.gif</ImageUrl>
      <NavigateUrl>http://www.163.com/</NavigateUrl>
      <AlternateText>链接到网易主页</AlternateText>
      <Keyword>网易</Keyword>
      <Impressions>25</Impressions>
   </Ad>
</Advertisements>
```

(3) 在 ASP.NET 页面中创建一个 AdRotator 服务器控件，将广告 XML 文件连接到这个控件。使用以下服务器控件标记来完成，示例代码如下：

```
<td>
   <asp:AdRotator
     ID="AdRotator1"
     runat="server"
     AdvertisementFile="~/XMLFile1.xml"/>
</td>
```

2.4.4　运行结果

至此，随机广告已制作完成，下面是整个随机广告的运行效果：右击 Random_Ads.aspx 文件，在弹出的快捷菜单中选择“设为起始页”命令，再单击工具栏中的“启动调试”按钮，Random_Ads.aspx 的显示效果如图 2-4 所示。



图 2-4 实现页面随机显示的效果

2.4.5 实例分析



源码解析：

该案例主要讲解的是 AdRotator 控件的应用，该控件的数据源是 XML 文件。AdRotator 控件是常用的广告图像控件，通过刷新页面，这个控件显示的图像是随机的，这是这个控件最大的优点，用起来也是很方便的。XML 文件保存所有图像的信息，但一定要符合 XML 文档规范。

2.5 编写节日提示日历

我们在博客或者是网站上会经常见到一个日历放置在博客的旁边，看上去既美观又实用。这一节我们就制作一个带节日提示的日历来讲解日历控件 Calendar，该控件显示一个日历。用户通过此日历可以定位到任意一年中的任意一天，或在不同的月份之间移动到任意日期，还可以配置日历以允许用户选择多个日期，包括整周或整月。



视频教学： 光盘/videos/2/Calendar.avi　　　　长度：4 分钟

2.5.1 基础知识——日历控件

Calendar 控件用于在浏览器中显示日历。该控件可显示某个月的日历，允许用户选择日期，也可以跳到前一个或下一个月。

接着我们来了解一下 Calendar 控件的属性，如表 2-1 所示。

表 2-1　Calendar控件的属性

属　性	说　明
Caption	日历的标题
CaptionAlign	日历标题文本的对齐方式
CellPadding	用于设置日历单元格边框与其内容之间的空白
CellSpacing	单元格之间的空白，以像素计
DayHeaderStyle	用于设置或返回一周中某天的部分的样式属性
DayNameFormat	用于设置日历中星期名称的格式
DayStyle	用于设置或返回日历中日期的样式
FirstDayOfWeek	用于规定日历中哪天是周的第一天
NextMonthText	用于规定日历中下一月的链接所显示的文本
NextPrevFormat	用于规定日历中下一月和上一月链接的格式
NextPrevStyle	用于设置和返回日历中下一月和上一月链接的样式
OtherMonthDayStyle	用于设置或返回日历中不属于当前月的日的样式
PrevMonthText	用于规定在日历中上一月的链接所显示的文本
SelectedDate	用于设置或返回日历中被选的日期
SelectedDates	用于设置或返回日历中被选的日期
SelectionMode	用于设置或返回用户如何选择日期
TodaysDate	用于设置或返回日历的当前日期
VisibleDate	用于设置或获取要在 Calendar 控件上显示的月份的日期

Calendar 控件的事件如表 2-2 所示。

表 2-2　Calendar控件的事件

事　件	说　明
OnDayRender	当每一天的单元格被创建时，所执行的函数的名称
OnSelectionChanged	当用户选择天、周或月时，所执行的函数名称
OnVisibleMonthChanged	当用户导航到不同的月时，所执行的函数名称

2.5.2　实例描述

日历，又叫挂历，即挂在墙上的日历，它记载着我们过去的和将要过的每一天，还记载着中国的传统节日。

以前做项目的时候，只用到过日历，但是还从来没有用过或者写过带节日提示的日历。本节我们就来共同研究这个带节日提示的日历是怎么写的。

2.5.3 实例应用

【例 2-5】编写节日提示的日历。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 JieRi_TiShi.aspx 页面的标题为“带节日的日历”。

(2) 在“工具箱”里找到 Calendar 日历控件，将其显示在页面中。在“属性”工具栏中找到 DayRender 事件，以鼠标双击，如图 2-5 所示。

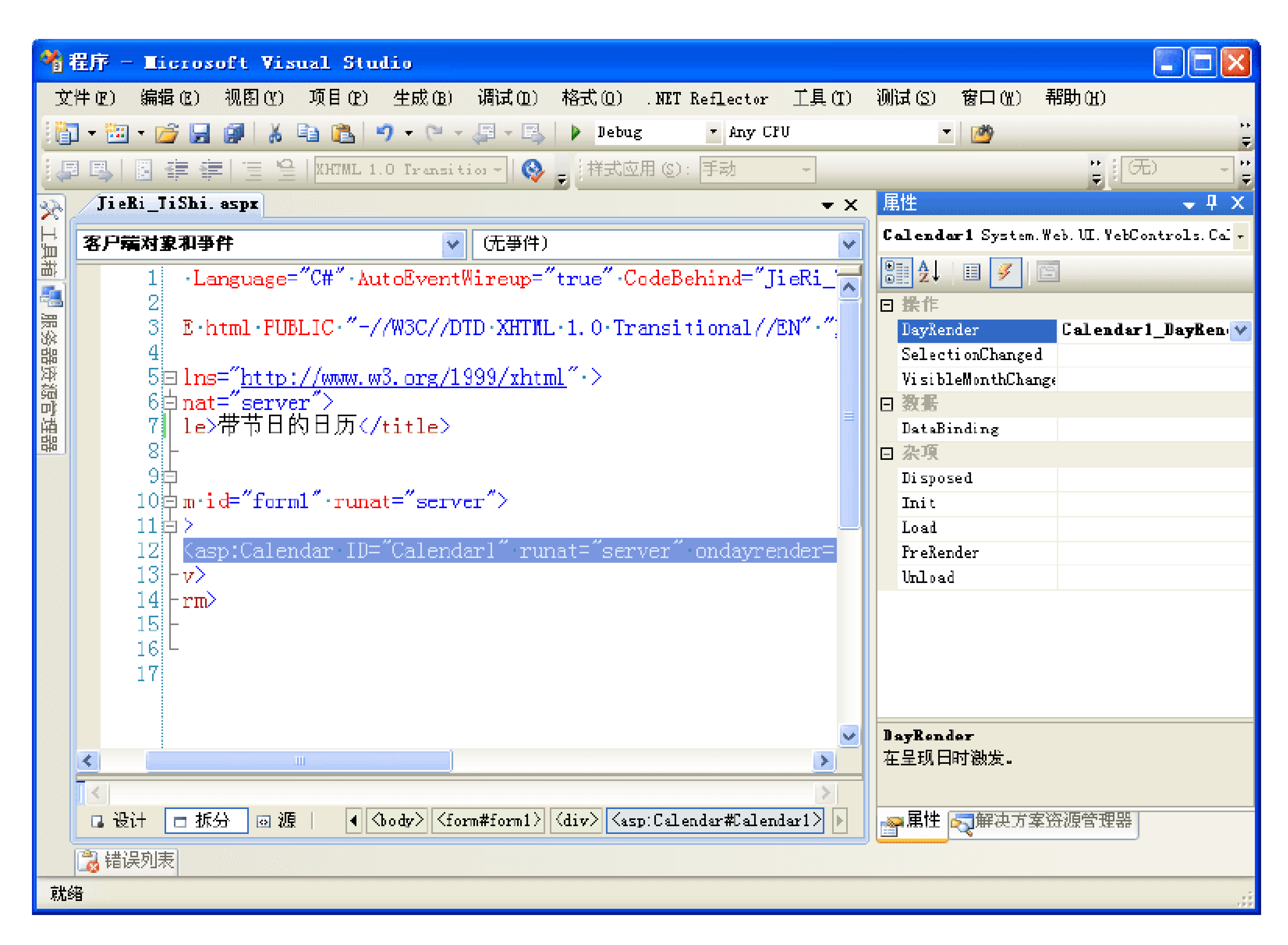


图 2-5 Calendar属性工具栏的DayRender事件

(3) 在后台 Calendar1_DayRender 事件里编写如下代码：

```
CalendarDay d = ((DayRenderEventArgs)e).Day;
TableCell c = ((DayRenderEventArgs)e).Cell;

if (e.Day.IsOtherMonth)
{
    e.Cell.Controls.Clear();
}
else
{
    try
    {
        string hol = holidays[e.Day.Date.Month][e.Day.Date.Day];
        if (hol != string.Empty)
        {
            e.Cell.Controls.Add(new LiteralControl(
              "<br><font color=blue size=2>" + hol + "</font>"));
        }
```

```
    }
    catch (Exception exc)
    {
        ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('"
          + exc.ToString() + "')</script>");
    }
}
```

(4)　在后台定义一个全局变量。代码如下：

```
String [][]holidays = new String[13][];
```

(5)　在 Page_Lode 中编写如下代码：

```
for (int i=0; i<13; i++)
{
    holidays[i] = new String[32];
}

holidays[1][1] = "元旦";
holidays[2][14] = "情人节";
holidays[3][8] = "妇女节";
holidays[3][12] = "植树节";
holidays[4][1] = "愚人节";
holidays[5][1] = "劳动节";
holidays[5][4] = "青年节";
holidays[5][12] = "护士节";
holidays[5][14] = "母亲节";
holidays[5][14] = "助残日";
holidays[6][1] = "国际儿童节";
holidays[6][5] = "环境保护日";
holidays[6][18] = "父亲节";
holidays[6][26] = "国际禁毒日";
holidays[7][1] = "中共诞辰";
holidays[8][1] = "建军节";
holidays[9][10] = "教师节";
holidays[10][1] = "国庆节";
holidays[11][23] = "感恩节";
holidays[12][1] = "艾滋病日";
holidays[12][12] = "西安事变";
holidays[12][25] = "圣诞节";
```

2.5.4　运行结果

按 Ctrl+F5 组合键运行程序，效果如图 2-6 所示。

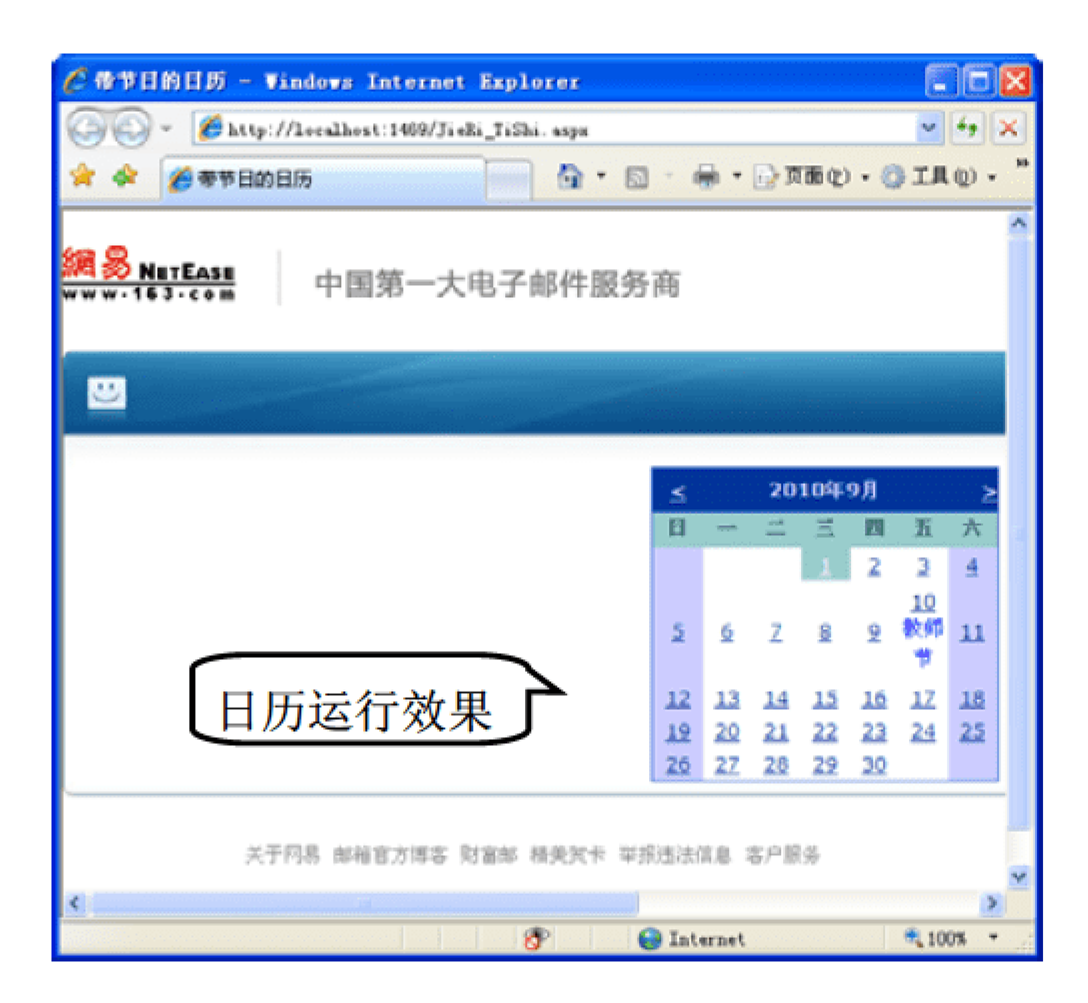


图 2-6　带节日提示的日历

2.5.5　实例分析



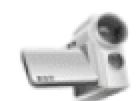
源码解析：

该案例的实现主要借助于 Calender 控件和 String 数组的应用，将节日提示的所有信息存放到了 String 数组里，当日历的每一天的单元格被创建时，将对应数组的值对添加到这个单元格中。

当然，如果这个单元格没有日期的话，要把这个异常信息处理掉。为了美观一点，可以将 Calendar 控件套用它自己现有的样式。

2.6　制作简历注册向导

ASP.NET 中 Wizard 控件简化了许多与生成以及收集用户输入的操作关联的任务。Wizard 控件比 MultiView 控件更富魅力。它同样支持每次显示几个视图中的一个，但它还包含一系列自定义的内建行为，包括导航按钮、带有分步链接的侧栏、样式和模板。



视频教学：光盘/videos/2/Wizard.avi　　　　长度：6 分钟

2.6.1　基础知识——注册控件

Wizard 控件是一个定义非常明确又很容易使用的控件，它可以引导用户输入参数或完成一次销售。当需要让用户按一组定义好的步骤操作时，Wizard 控件是最好的选择。

通常，向导表示一个任务，用户在其间进行线性移动，从当前步骤前进到下一步(或者在做更正时退回到上一步)。ASP.NET 的 Wizard 控件还支持非线性导航，也就是说，它允许你根据用户提供的信息忽略某些步骤。

默认情况下，Wizard 控件提供导航按钮并在左边提供一个带有每个步骤链接的侧栏。设置 Wizard.DisplaySideBar 属性为 false 可以隐藏侧栏。一般情况下，当需要强制线性导航并阻止用户跳离固定的次序时，会用到这样的设置。你可以用任意的 HTML 控件或 ASP.NET 控件来提供每一步骤的内容。下面我们来简单了解一下 Wizard 控件的几个属性。

- Title：步骤的描述性名称。这个名称用在侧栏作为链接显示的文字。
- StepType：步骤的类型，它的值来自 WizardStepType 枚举。这个值确定要为这个步骤显示的导航按钮的类型。可选项包括 Start(显示 Next 按钮)、Step(显示 Next 和 Previous 按钮)、Finish(显示 Finish 和 Previous 按钮)、Complete(不显示按钮，如果启用了侧栏也会把它隐藏)、Auto(步骤的类型按它在集合中的位置推断)。默认值是 Auto，它表示第一个步骤是 Start，最后一个步骤是 Finish，所有其他步骤是 Step。
- AllowReturn：表示用户是否可以重新回到这一步。如果为 false，用户通过这一步之后，就再也不能返回这里。侧栏的链接对这个步骤不起作用，它的下一个步骤的 Previous 按钮不是跳过这一步就是彻底隐藏(依赖于上一步骤设置的 AllowReturn 值)。

2.6.2　实例描述

现在互联网的技术越来越发达，范围越来越广，连老爷爷老奶奶都会在网上聊天了。不管是用什么样的聊天工具，如 MSN、QQ、飞信等，首先就要把它安装到我们的电脑上，这是很容易的，只要遵照指示，单击“下一步”、“下一步”……“完成”等按钮即可。在这一节中，我们将要讲到的就是这个有“下一步”的东西，也就是 Wizard 控件，又叫向导控件。

2.6.3　实例应用

【例 2-6】制作简历注册向导。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 jianli_xiangdao.aspx 页面标题为“简历注册向导”。

(2) 添加一个 Wizard 控件，在“属性”窗口设置属性 width 的值为 504px 和 height 值为 150px。在设计视图单击 Wizard 控件右上角的“小三角”，然后打开 WizardStep 编辑器对 WizardStep 编辑、设计我们所需要设计的步骤，即“个人信息”、“教育背景”、“工作经验”、“应聘职位”4 步。如图 2-7 所示。



图 2-7　WizardStep集合编辑器

(3) 为了控件 Wizard 的美观性，可以在设计视图单击 Wizard 控件右上角的“小三角”，然后单击“自动套用格式”，这里选择了“传统型”，如图 2-8 所示。

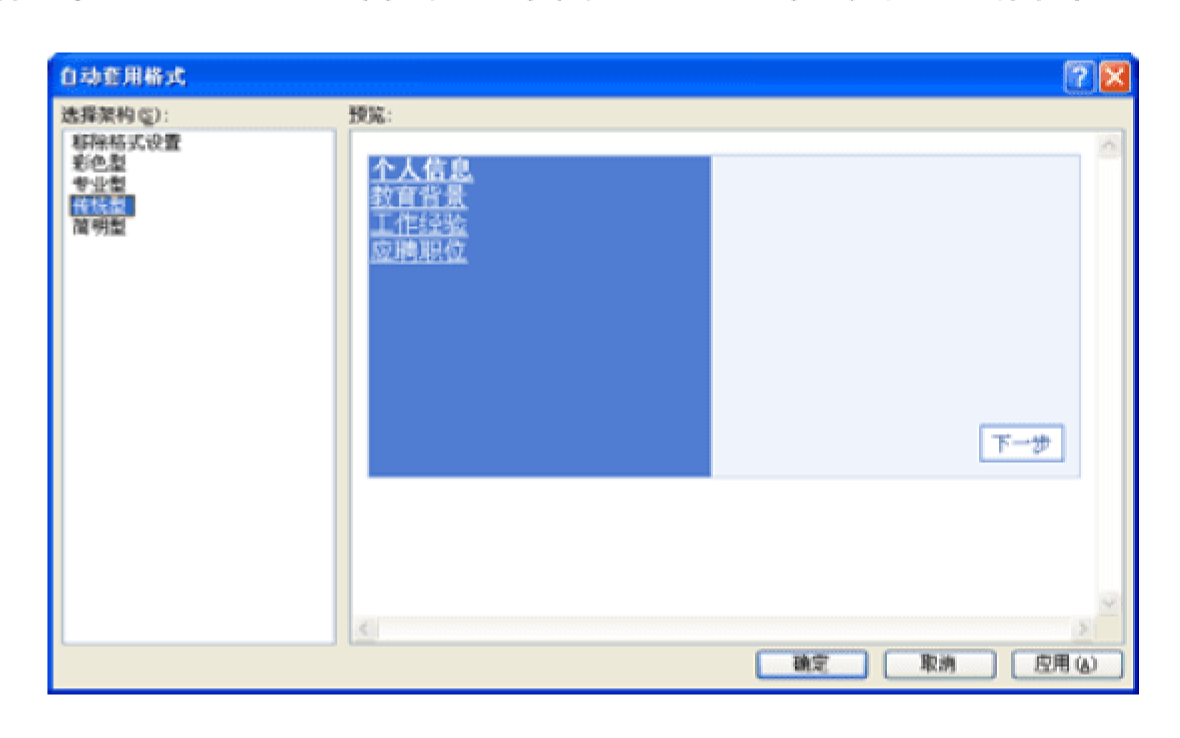


图 2-8 自动套用格式

(4) 设计好向导步骤后，接下来就应该设计每个步骤的内容了，首先设计第一个步骤的内容页，在内容页添加一个 3 行 4 列的表格，从第一行第一列往下依次插入文本“姓名”、“身份证号”、“手机号码”，从第一行第三列往下依次插入文本“出生年月”、“籍贯”、“现居住地”，然后可以在设计视图中调节表格的宽度和高度，如图 2-9 所示。

(5) 在“教育背景”内容页添加一行四列的表格，在第一行第一列和第三列分别插入文本“毕业院校”、“所学专业”，在第一行第二列和第四列分别插入文本框，如图 2-10 所示。



图 2-9 “个人信息”内容模板设计



图 2-10 “教育背景”内容模板设计

(6) 在“工作经验”内容页添加一个文本域，如图 2-11 所示。

(7) 在“应聘职位”内容页添加一个一行 4 列的表格，如图 2-12 所示。

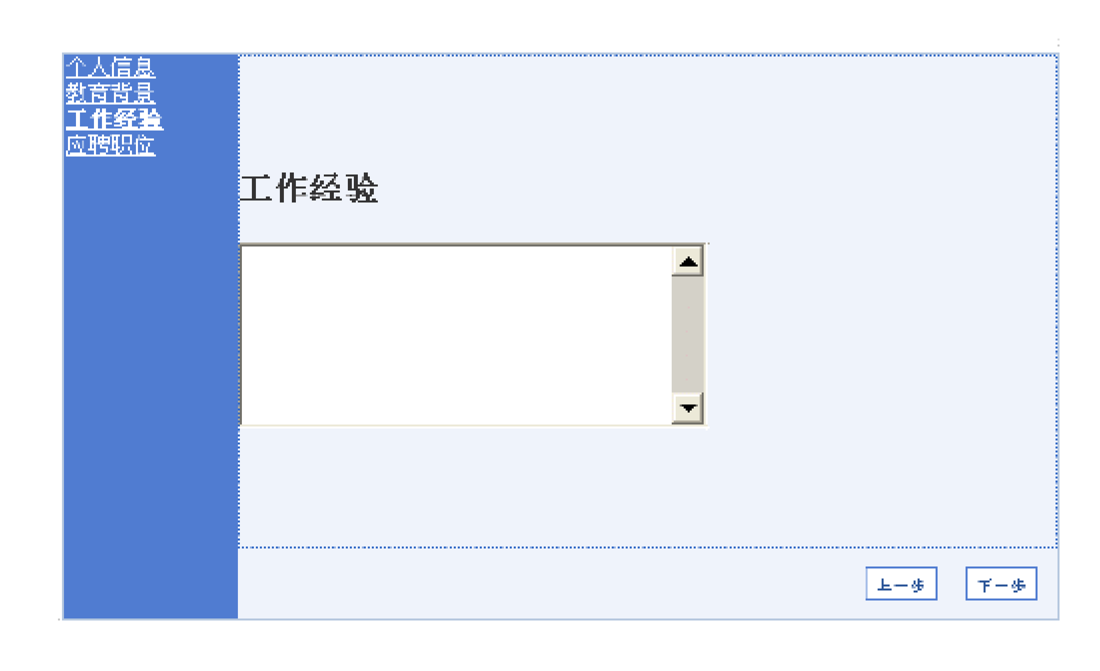


图 2-11 “工作经验”内容模板设计

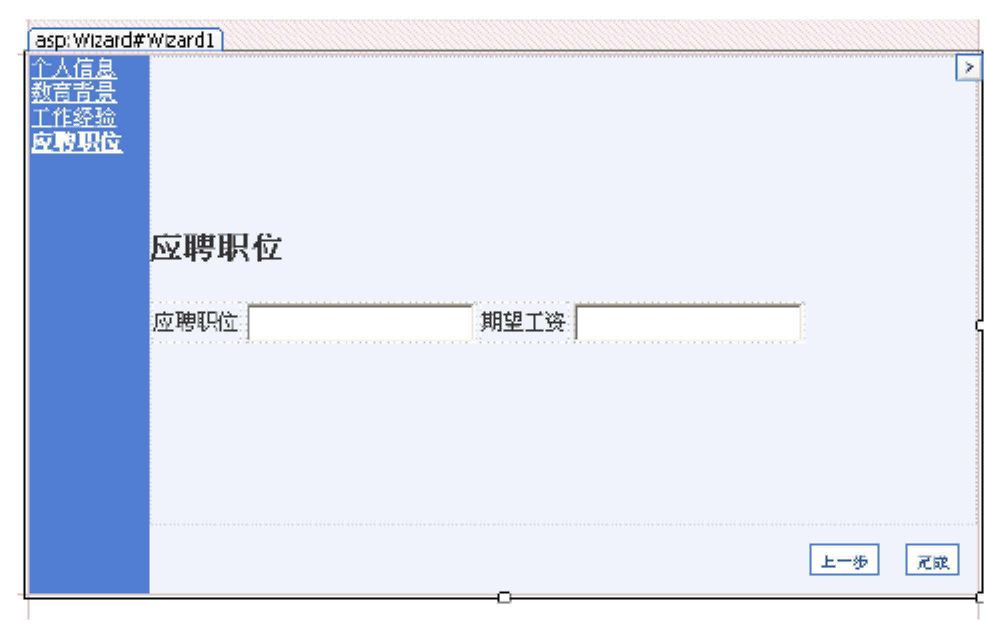


图 2-12 “应聘职位”内容模板设计

2.6.4　运行结果

在以上控件添加完成的情况下，整个页面的布局也就完成了，如图 2-13 所示为最终页面的效果。



图 2-13　简历注册向导

2.6.5　实例分析



源码解析：

简历注册向导就相当于网页上的用户注册。该案例主要介绍了向导控件——Wizard 控件，使用多个步骤来描绘用户数据输入的不同部分。Wizard 控件不仅有助于管理所要显示步骤，而且有助于维护所收集的数据。简历注册向导包括“个人信息”、“教育背景”、“工作经验”、“应聘职位”这 4 个步骤，填写每一步的相关信息，将完成简历注册。

2.7　使用用户控件为网站设计导航系统

网站的导航系统可以有很多种实现方法，如 Table 表格、TreeView 控件、ul 列表、Menu 控件等，一般情况下，都是直接写在页面中的，在本节中，我们将它们封装到了用户控件里。用户控件使开发人员能够根据应用程序的需求，方便地定义和编写控件。开发所使用的编程技术将与编写 Web 窗体的技术相同，只要开发人员对控件进行修改，就可以将使用该控件的页面的所有控件都进行修改，为了确保用户控件不会被修改、下载，被当成一个独立的 Web 窗体来运行，用户控件的后缀名为“.ascx”，它的基类是 UserControl。当用户访问页面时，用户控件是不能直接被用户访问的。



视频教学：光盘/videos/2/用户控件.avi 长度：10 分钟

2.7.1 基础知识——用户控件

一个 Web 用户控件与一个完整的 Web 窗体页相似，它们都包含一个用户界面页和一个代码隐藏文件。用户控件在以下方面与.aspx 文件不同：

- 扩展名必须为.ascx。
- 用户控件中不包含<html>、<body>和<form>元素。
- 真正使用时把用户控件放在 aspx 文件中使用，而 aspx 文件是有<html><body>这些元素的，不能重复。

2.7.2 实例描述

有了 ASP.NET，我们拥有了一个新的简单的工具来编写可重用的代码——用户控件。用户控件提供了这样一种机制，它使得我们可以建立能够非常容易被 ASP.NET 页面使用或者重新利用的代码部件。用户控件，用简单易懂的语言来描述，就像我们买的自行车一样，想骑或者需要骑的时候都可以驾驭它。

下面我们就用一个网站的导航系统来讲解一下 Web 用户控件。

2.7.3 实例应用

【例 2-7】使用用户控件为网站设计导航系统。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 daohang.aspx 页面标题为“使用用户控件为网站设计导航系统”。

(2) 在这个项目里添加一个 Web 用户控件，命名为“DaoHang.ascx”。然后设计导航菜单，我们这里用了列表来显示导航。示例代码如下：

```
<div id="nav">
<ul>
    <li><a href="#">新闻</a></li>
    <li><a href="#">网页</a></li>
    <li><a href="#">音乐</a></li>
    <li><a href="#">说吧</a></li>
    <li><a href="#">知道</a></li>
    <li id="special"><a href="#">地图</a></li>
</ul> 
</div>
```

(3) 为了导航的美观性，我们给它添加了些样式。这些样式可以写到 Web 用户控件中，也可以单独的放到 CSS 文件中。示例代码如下：

```
<style type="text/css">
#nav {
```

```
    margin: 0;
    background: #669900;
    width: 100%;
    border-bottom: 1px solid #006600;
    border-top: 1px solid #006600;
    border-left: none;
    border-right: 1px solid #006600;
    height: 20px;
}
</style>
```

(4) 在 daohang.aspx 页面中调用用户控件，首先就要先在页面注册 DaoHang.ascx 用户控件。示例代码如下：

```
<%@ Register Src="~/DaoHang.ascx" TagName="daohang" TagPrefix="uc2"%>
```

然后在页面中使用。示例代码如下：

```
<uc2:daohang ID="dh" runat="server" />
```

至此，程序编写完成。

2.7.4　运行结果

Web 用户控件不能直接运行，必须在网页中才能运行。我们将 daohang.aspx 页面设为启动项，然后运行它。运行效果如图 2-14 所示。



图 2-14　使用用户控件为网站设计导航系统

2.7.5　实例分析

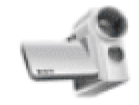


源码解析：

该案例主要讲解了 Web 用户控件的编写及使用。Web 用户控件的后缀名为“.ascx”，在它里面可以使用任何 HTML 控件和服务器控件，也可以添加 JavaScript 脚本和 CSS 样式。在本案例中主要添加了样式，用的是 HTML 控件，显示的效果与普通的 aspx 页面是一样的，只不过用户控件可以在多个页面中使用。

2.8　在网页中实现树形导航

树形导航可以带你随意浏览任何一个网页。在本节中，我们就要用到这样一个控件了——TreeView 控件。它是导航控件，与它并列的还有 Menu 控件和 SiteMapPath 控件。



视频教学：光盘/videos/2/TreeView.avi　　　　长度：3 分钟

2.8.1　基础知识——TreeView控件

TreeView 控件又称为树视图控件，它能够以层次或树形结构显示数据，并提供导航到指定页面的功能。它支持以下 7 个常用的功能：①站点导航，即导航到其他页面的功能；②以文本或链接方式显示节点的内容；③可以将样式或主题应用到控件及其节点；④数据绑定，允许直接将控件的节点绑定到 XML、表格或关系数据源；⑤可以为节点实现客户端功能；⑥可以在每一个节点旁边显示复选框按钮；⑦可以使用编程方式动态设置控件的属性。

TreeView 控件由节点组成，节点的类型为 TreeNode，且这些节点存在一定的层次关系。TreeNode 提供了许多常用的属性，如表示节点文本的 Text 属性、表示节点值的 Value 属性等，具体说明如表 2-3 所示。

表 2-3　TreeNode类的常用属性表

属　性	描　述	属　性	描　述
Text	节点的文本	Checked	表示节点的复选框是否被选中
Value	节点的值	Selected	表示是否选择了该节点
Depth	节点的深度	Expanded	表示是否扩展了该节点
Parent	节点的父节点	ImageUrl	节点显示图像的 URL
ChildNodes	节点的第一级子节点	NavigateUrl	节点导航的 URL
ShowCheckBox	表示是否显示复选框	Target	节点导航到的目标窗口或框架

TreeView 控件不仅仅只具有导航的功能，它还能够处理页面回发的数据。TreeView 有一个 Value 属性，可以作为页面数据传递的载体。SelectedValue 向外公开了当前选择节点的 value

值。SelectedNode.ValuePath 属性可以获取当前节点在层次结构里的等级。

2.8.2　实例描述

做项目的时候，经常会用到导航这种东西，它是整个系统的核心，也是关键。导航功能有很多种实现方法，比如说站点地图、Menu 控件、Table 表格等。作者一般都用表格或者列表，纯属个人习惯，可以模仿。

所谓树形导航，是一种结构类似于大树的东西，而且名字里也有树的英文拼写，就是 TreeView 控件。本节我们就来共同探讨一下如何使用 TreeView 控件。

2.8.3　实例应用

【例 2-8】在网页中实现树形导航。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 treeview.aspx 页面标题为“在网页中实现树形导航”。

(2) 在网页中添加一个 TreeView 控件，在 Nodes 节点下添加子节点，然后在设计视图中给 TreeView 控件自动套用格式，这里使用的是“新闻型”。示例代码如下：

```
<asp:TreeView ID="TreeView1" runat="server" ImageSet="News" NodeIndent="10">
    <ParentNodeStyle Font-Bold="False" />
    <HoverNodeStyle Font-Underline="True" />
    <SelectedNodeStyle Font-Underline="True" HorizontalPadding="0px"
      VerticalPadding="0px" />
    <Nodes>
        <asp:TreeNode Text="新闻" NavigateUrl="#">
            <asp:TreeNode Text="焦点新闻" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="国内" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="国际" NavigateUrl="#"></asp:TreeNode>
        </asp:TreeNode>
        <asp:TreeNode Text="互联网" NavigateUrl="#">
            <asp:TreeNode Text="搜索引擎" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="电子商务" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="网络游戏" NavigateUrl="#"></asp:TreeNode>
        </asp:TreeNode>
        <asp:TreeNode Text="财经" NavigateUrl="#">
            <asp:TreeNode Text="股票" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="理财" NavigateUrl="#"></asp:TreeNode>
            <asp:TreeNode Text="经济民生" NavigateUrl="#"></asp:TreeNode>
        </asp:TreeNode>
    </Nodes>
    <NodeStyle Font-Names="Arial" Font-Size="10pt" ForeColor="Black"
      HorizontalPadding="5px" NodeSpacing="0px" VerticalPadding="0px" />
</asp:TreeView>
```

2.8.4 运行结果

运行程序，效果如图 2-15 所示。



图 2-15 在网页中实现树形导航

2.8.5 实例分析



源码解析：

该案例主要讲解了导航控件中一种——TreeView 控件，它也是比较典型的树形控件。TreeView 控件允许您向树结构中添加多个根节点，在 Nodes 节点下添加 TreeNode 子节点，TreeNode 节点下还可以添加 TreeNode 子节点，可以无限级别添加子节点。

2.9 实现导航路径

任何由多个页面组成的网站都需要某种导航用户接口。所谓站点导航其实就是让浏览者能够知道自己在 Web 海洋里所处的位置，并能够快速返回原来的位置或者快速找到你想要进入的页面。

 视频教学：光盘/videos/2/SiteMapPath.avi　　长度：5 分钟

2.9.1 基础知识——SiteMapPath控件

SiteMapPath 控件提供导航路径，也就是说，它显示用户当前位置并允许用户使用链接回

到更高的层级。SiteMapPath 是 ASP.NET 包含在导航控件中的一种，另外两个导航控件是 Menu 控件和 TreeView 控件。

创建一个.sitemap 文件，其实就是一个 XML 文件，包括有着层次结构的<siteMapNode>元素。<siteMapNode>元素的属性如表 2-4 所示。

表 2-4　siteMapNode元素的属性

属　性	说　明
Url	链接地址
Title	显示的标题
Description	描述(ToolTip)
resourceKey	本地化用的(要在<siteMap>节点上加上这个属性 enableLocalization=true)
securityTrimmingEnabled	是否让 sitemap 支持安全特性
roles	哪些角色可以访问当前节点，多角色用逗号隔开(需要将 securityTrimmingEnabled 设置为 true)
siteMapFile	引用另一个 sitemap 文件



应用权限的时候，Web.config 文件中的 SiteMap 节点的 Provider 也要有相对应的配置(securityTrimmingEnabled="true")。

SiteMapPath 控件常见的属性如下。

- ShowToolTips：取值为 true 或 false。当设置 ShowToolTips 为 true 时(默认值)，将显示在 web.sitemap 里由 Description 属性描述的文本信息，反之则不显示。
- PathDirection：取值为 RootToCurrent 或 CurrentToRoot，默认值为 RootToCurrent，显示路径为从高级(Root)到当前级别(Current)；如果设置为 CurrentToRoot，则显示为从当前级别到最高级别。
- ParentLevelsDisplayed：设置显示的父级等级，默认值为-1，表示全部显示。如果值为 2，则显示两级父级目录。
- PathSeparator：默认站点导航是以“>”分开的，你可以使用自定义父/子路径之间的分割符号。例如 PathSeparator= “>>”。
- RenderCurrentNodeAsLink：取值为 true 或 false。默认值为 false，表示当前页面显示为纯文本样式；如果值为 true，则显示为链接的样式。

SiteMapPath 显示的每个节点都是 HyperLink 或 Literal 控件，可以将模板或样式应用到这两种控件。

2.9.2　实例描述

传统上，一个简单的导航可能仅仅是一个静态链接，但是随着 Web 应用程序的不断扩大，静态导航越来越满足不了应用的需求，因此开发人员不得不自己或者通过第三方组件来构建复杂的导航系统。幸运的是 ASP.NET 内置了导航系统功能。

本节中，我们就来了解一下如何制作导航路径。

2.9.3 实例应用

【例 2-9】实现导航路径。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 daohang_url.aspx 页面标题为“实现导航路径”。

(2) SiteMapPath 控件和 Menu/TreeView 控件不同，显示的数据并不是由 SiteMapProvider 提供的，由于它不支持 SiteMapDataSource 控件属性，所以该控件不能够绑定在具有分层结构的控件上。既然 SiteMapPath 的数据源默认不是来自数据绑定上，那么它来自哪里呢？

这里就需要利用 ASP.NET 提供的文件 web.sitemap(站点地图)。在这个网站中添加新建项，找到站点地图文件，默认文件名为 Web.sitemap。然后在文件中按层次添加节点。代码如下：

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
<siteMapNode url="~/jianli_xiangdao.aspx" title="首页" description="首页描述">
<siteMapNode url="~/Login.aspx" title="频道 1" description="频道 1 描述" />
<siteMapNode url="~/daohang_url.aspx" title="频道 2" description="频道 2 描述" />
</siteMapNode>
</siteMap>
```

(3) 最后在 jianli_xiangdao.aspx、Login.aspx、daohang_url.aspx 页面中添加 SiteMapPath 控件，设置它的 RenderCurrentNodeAsLink 为 true。示例代码如下：

```
<asp:SiteMapPath ID="SiteMapPath1" RenderCurrentNodeAsLink="true"
  ShowToolTips="true" runat="server" Font-Names="Verdana" Font-Size="12"
  PathSeparator="<<">
</asp:SiteMapPath>
```

2.9.4 运行结果

以上步骤已完成，下面我们来运行看看，如图 2-16 所示。



图 2-16 实现导航路径

2.9.5　实例分析

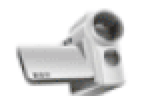


源码解析：

在该案例中，站点导航主要围绕两个重点来实现——SiteMapPath 控件和 Web.sitemap 文件。这里我们需要掌握的是 SiteMapPath 控件必须在使用到它的页面出现，然后就是如何编写 Web.sitemap 文件的内容。

2.10　在网页中实现皮肤切换

如何在网页中实现皮肤切换呢？在 ASP.NET 中，要想换肤，就要用到皮肤控件和主题了，在应用主题的同时如何使同一类型的控件在应用程序的不同部分具有不同的外观呢，本节将会具体介绍。



视频教学： 光盘/videos/2/主题.avi　　　　长度：7 分钟

2.10.1　基础知识——主题

主题可以包括外观文件(.skin)和 CSS 样式文件。你可以建立任意多个样式或者外观文件。每一个外观文件都可以包含一个或多个控件外观的定义。而控件外观定义就是指定控件的外观属性。

1. 全局主题和应用程序主题

主题包括全局主题和应用程序主题。应用程序主题放在应用程序根目录下的 App_Themes 目录中；全局主题放在 ASP.NET 安装目录下的 ASP.NETClientFiles 文件夹下的 Themes 目录中，例如%WINDIR%\Microsoft.NET\FrameWork\<version>\ASP.NETClientFiles\Themes。对于 IIS 网站，全局主题的位置是 Inetpub\wwwroot\aspnet_client\system_web\<version>\Themes。

2. 为页面指定主题

为了使用外观文件，需要在页面里添加对主题的引用，可以有多种方式引用主题，基本方式如下：

```
<%@ Page Language="C#" Theme="Blue" %>
```

3. 在web.config文件中指定主题

在 web.config 的<page theme="…"/>节中，也可以为应用程序中的所有页定义应用的主题。如果要对特定页取消设置此主题，可以将 Page 指令的 Theme 属性设置为空字符串。



母版页不能应用主题，而应在内容页或配置(web.config 文件)中设置主题。

下面的示例代码将当前应用程序的所有页指定同一个主题：

```
<system.web>
<pages theme="blue"></pages>
</system.web>
```

4. 对控件禁用主题

通过将 EnableTheming 属性设置为 false，可将特定控件排除在外，使其属性不会被主题重写。下面的示例代码中，由于 Label2 将其 EnableTheming 属性值设置为了 false，所以它将不继承外观文件里定义的样式。示例代码如下：

```
<asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
<asp:Label ID="Label2" runat="server" Text="Label" EnableTheming="false">
</asp:Label>
```

5. 默认外观与命名外观

默认情况下，外观文件中的控件定义应用于受主题影响的应用程序的页中同一类型的所有控件。但是，我们可能希望同一类型的控件在应用程序的不同部分中具有不同的外观。可能希望 TextBox 控件的边框颜色不一样。在主题中使用命名外观可以实现此功能。

通过创建不同的控件定义，可以在外观文件中为同一类型的控件定义不同的样式。可以将这些控件定义的某个单独的 SkinID 属性设置为所选择的名称，然后对页中要应用此特定外观的控件设置此 SkinID 值。示例代码如下：

```
<asp:TextBox ID="TextBox1" runat="server" SkinID="Blue"></asp:TextBox>
```



在 Skin 文件中只可以设置控件的样式。

2.10.2 实例描述

同一款手机可以有不同的颜色，同一件衣服可以有不同的颜色，同一款车可以有不同的颜色。那同一个网站呢，当然也是可以的了，同一个网站可以有不同的皮肤显示。以前做的网站都是选好颜色、样式，然后才开始做，这样，整个网站下来，就只有一种皮肤。其实为了让用户能够一饱眼福，我们也可以给它换肤啊。

下面就一起来学习如何换肤。

2.10.3 实例应用

【例 2-10】在网页中实现皮肤切换。

(1) 打开 Visual Studio 2008，新建一个 ASP.NET 网站，并设置 ThemePage.aspx 页面标题为“在网页中实现皮肤切换”。

(2) 在页面中添加一个下拉控件。添加三个选项，“默认”、“黄皮肤”、“蓝皮肤”，并且设置 AutoPostBack 为 true。示例代码如下：

```
<asp:DropDownList ID="DropDownList1" runat="server" AutoPostBack="true"
  onselectedindexchanged="DropDownList1_SelectedIndexChanged" >
    <asp:ListItem Value="" Text="默认"></asp:ListItem>
    <asp:ListItem Value="Yellow" Text="黄皮肤"></asp:ListItem>
    <asp:ListItem Value="Blue" Text="蓝皮肤"></asp:ListItem>
</asp:DropDownList>
```

(3) 然后添加两个外观文件，默认在 App_Themes 目录下。分别命名为 Blue 和 Yellow，然后分别添加两个样式表，命名 Blue 和 Yellow。

在两个样式表里添加样式，示例代码如下：

```
.aa
{
    background-color:Yellow;
    color:Blue;
}
```

(4) 将主题应用到页面中。后台代码为：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Request.QueryString["t"] != null)
        {
            DropDownList1.SelectedValue = Request.QueryString["t"].ToString();
        }
    }
}
protected void Page_PreInit(object sender, EventArgs e)
{
    string theme = Request.QueryString["themes"];
    Page.Theme = theme;
}
protected void DropDownList1 SelectedIndexChanged(object sender, EventArgs e)
{
    Session["theme"] = DropDownList1.SelectedItem.Value;
    Response.Redirect("ThemePage.aspx?themes="+Session["theme"].ToString());
}
```

2.10.4　运行结果

当选择下拉列表中的任何一个选项，网页就会呈现不同的颜色，如图 2-17 所示。



图 2-17 在网页中实现皮肤切换

2.10.5 实例分析



源码解析：

在网站添加了两个主题——Blue 和 Yellow，它们里面也包含了 Blue 和 Yellow 两个样式文件，所谓的换肤，就是实现了主题在页中的应用。外观文件在里面起到的作用就是可以为所有同一类型的控件定义样式，而样式文件则可以定义页面的样式。

2.11 常见问题解答

2.11.1 在HTML控件Input中把浏览按钮的背景更换成图片



ASP.NET 中，HTML 控件 Input(File)如何把浏览按钮的背景更换成图片？
网络课堂：http://bbs.itzcn.com/thread-2811-1-1.html

在 ASP.NET 中，HTML 控件 Input(File)如何把浏览按钮的背景更换成图片(如 ImageButton 一样)？

【解决办法】：使用 CSS 样式来控制就可以了。可以这样写：

```
<input id="HTML_Browse_up" type="file" runat="server" name="HTML_Browse_up"
  style="background-image:url(../images/css_tutorials/background.jpg)"/>
```

不过这里要注意图片的路径，这里是相对路径！祝你成功啊！

2.11.2　为什么我的CheckBoxList获得的选中状态不正确



为什么我的 CheckBoxList 获得的选中状态不正确？
网络课堂：http://bbs.itzcn.com/thread-2823-1-1.html

前台有一个 CheckBoxList 控件，代码如下所示：

```
<asp:CheckBoxList
  ID="cblPopedoms"
  runat="server"
  RepeatColumns="5"
  RepeatDirection="Horizontal"
  RepeatLayout="Flow"
  DataValueField="Pid"
  DataTextField="Pname">

</asp:CheckBoxList>
```

利用绑定事件设置它所有 ListItem 的 Selected=True，然后自己在页面中把几个复选框取消选中。在提交表单的时候获得所有的 ListItem 的 Selected 属性仍然都是 True，这是为什么啊？是需要用 SelectedIndexChanged 事件来动态改变 Selected 的值吗？

【解决办法】：呵呵……不用的。看看你是否在 if(!IsPostBack)里面？也就是说如果没有在 if(!IsPostBack)里面执行程序的话，那么程序将会重新加载，所以在提交表单的时候获得所有的 ListItem 的 Selected 属性仍然都是 True。

2.11.3　如何禁止选择Calendar控件中已经过去的时间



如何禁止选择 Calendar 控件中已经过去的时间？
网络课堂：http://bbs.itzcn.com/thread-2887-1-1.html

比如今天是 2010 年 12 月 9 日，则在 Calendar 控件中显示的时间，比 2010 年 12 月 9 日早的时间都不能选择，比如 2010 年 12 月 8 日；程序语言 C#；VS2008 中能用。

这不难，用下面的代码就搞定了：

```
protected void Calendar1_DayRender(object sender, DayRenderEventArgs e)
{
    if (e.Day.Date < DateTime.Now.Date)
    {
        e.Day.IsSelectable = false;
    }
}
```

首先判断 calendar 控件中显示的时间是不是早于当前时间，如果早于当前时间，则将显示的时间的 IsSelectable 属性设置为 false 即可。

2.12 习　　题

一、填空题

(1) 表单提供了__________和 POST 两种数据传输方式。

(2) __________和 Text 属性都可以指定验证控件的错误提示消息。Text 属性的优先级高于__________属性。如果同时设置了这两个属性的值，则验证控件优先显示__________属性的值。

(3) XML 文档结构可以分为三个部分：序言、__________和尾声。

(4) 用户控件的后缀名是__________，它的基类是__________。

(5) ASP.NET 提供的导航控件有 Menu、____________和____________。

二、选择题

(1) Display 属性指定控件错误消息的显示方式。以下选项__________是它的属性值。

A. None

B. Dynamic

C. Static

D. Both

(2) 验证邮箱是否有效的正则表达式是__________。

A. \w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

B. \w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)

C. \w+([-+.']\w+)*@+([-.]\w+)*\.\w+([-.]\w+)*

D. \([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*

(3) ASP.NET 中的向导控件是__________。

A. Wizard

B. DropDownList

C. Menu

D. TreeView

(4) 外观文件的后缀名是__________。

A. cs

B. theme

C. skin

D. aspx

(5) 对控件禁用主题的属性是__________。

A. Enable

B. Enabling

C. Theme

D. EnableTheming

三、上机练习

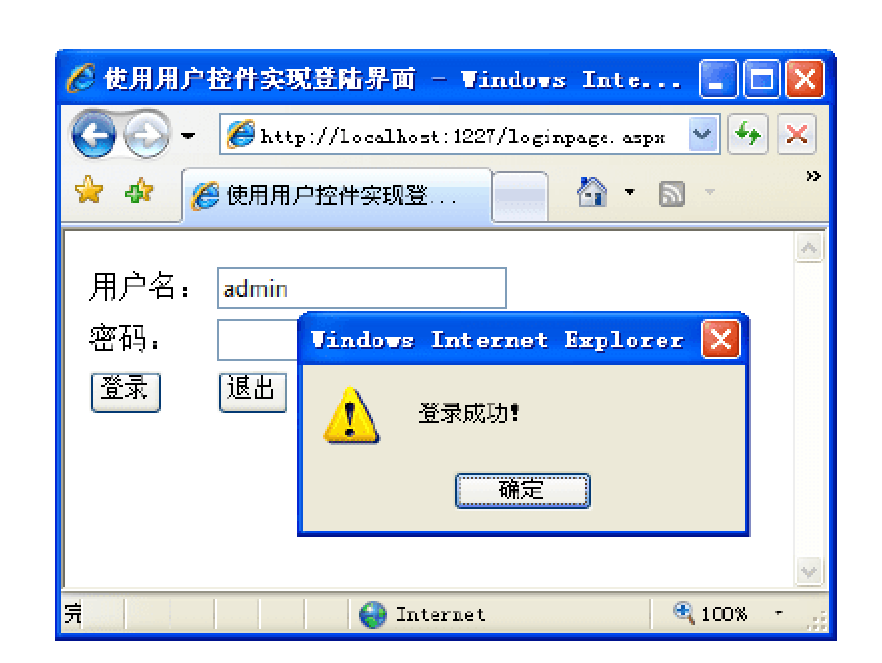
上机练习：使用用户控件制作简单的登录。

要求实现带验证功能的用户登录。

(1) 判断用户是否登录成功。

(2) 当单击“退出”按钮时，关闭网页。

效果如图 2-18 所示。



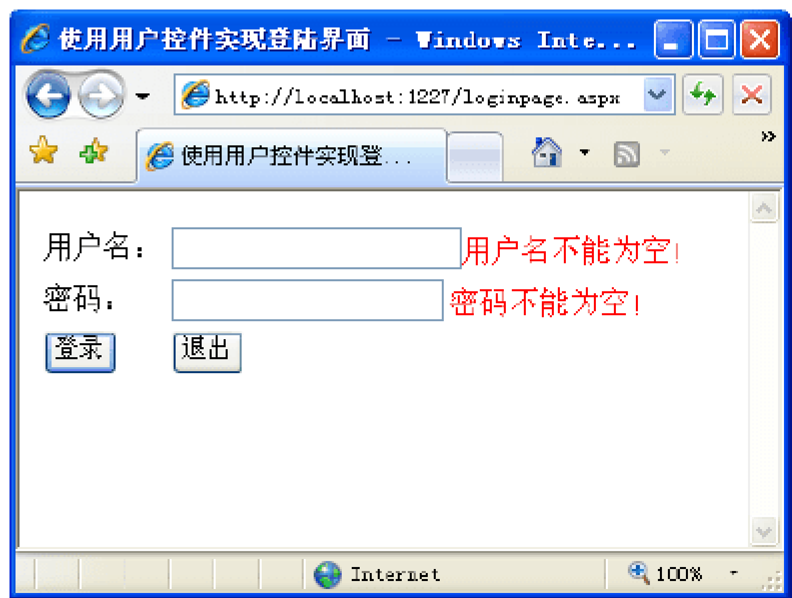


图 2-18　带验证的用户登录



第 3 章　剖析ASP.NET页面机制

内容摘要：

尽管 ASP.NET 中采用的是事件响应模式，使程序开发人员和最终用户感觉与 WinForm 程序非常接近，但是它毕竟还是 Web 应用程序。而 Web 应用程序的特点，就是基于浏览器与服务器的请求与响应的执行方式。所以无论 ASP.NET 最终如何对用户体验进行封装，它都无法脱离最基本的 B/S 结构的程序运行原理，用户在 Web 页面做的任何类似 WinForm 程序一样需要服务器响应的操作最终都将以传统的 Post 方式提交到服务器，而服务器就根据页面状态信息处理并响应页面请求。

所以，虽然 ASP.NET 改变了传统的 Web 开发的很多习惯，但是，我们在进行 Web 开发的时候还是不可避免地要用到请求(Request)、响应(Response)、会话(Session)、应用程序(Application)等传统的服务器端辅助对象。同时为了实现好的用户体验，ASP.NET 还加入了一些独有的辅助对象，如页面(Page)、视图状态(ViewState)、服务器实用工具(Server)等对象。

学习目标：

- 了解 ASP.NET 文件
- 了解 ASP.NET 页面指令
- 熟练使用 Request 对象
- 熟练使用 Response 对象
- 熟练使用 Session 对象
- 熟练使用 Server 对象
- 熟练使用 Application 对象
- 熟练使用 Page 对象

3.1　你所了解的ASP.NET文件扩展名

文件是操作系统里独立的最小存储单位。文件有许多类型，称为文件格式。扩展名就是操作系统用来标识文件格式的一种机制。

在传统的 Web 应用程序中，除了常用的资源型文件(如图片、Flash、数据库等)和静态的 Web 页面，一种语言只定义自己特有的一种或两种文件格式。比如 ASP 程序一般扩展名为.asp，可能的话会定义一些扩展名为.inc 的被包含文件。但在 ASP.NET 应用程序中，却提供了很多种类型的文件。

打开一个 ASP.NET 项目，经常会看到以下多种类型的文件，如表 3-1 所示。

表 3-1　ASP.NET文件扩展名

扩 展 名	说　明
.sln	解决方案文件，为解决方案资源管理器提供显示管理文件的图形接口所需的信息，包括项目结点及项目路径等
.csproj	项目文件，记录应用程序所需的引用、数据连接、文件夹和文件的信息
.aspx	Web 窗体，由两部分组成：页面视图和后台程序代码。页面视图包括(HTML 结构、服务器端控件和静态文本)，后台程序代码存储页面项目的程序逻辑。ASP.NET 将这两个组成部分分别存储在一个独立的文件中。.aspx 文件存储的就是页面视图信息
.cs	C#代码文件，存储所有程序业务逻辑的 C#代码
.aspx.cs	Web 窗体的后台程序代码文件，它本也属于.cs 文件，只不过它专门用来处理相应的.aspx 视图文件的展示逻辑
.ascx	用户控件，用户自定义的页面控件文件
.asmx	asmx 文件包含 WebService 处理指令，并用作 Web Services 的可寻址入口点
.asax	Global.asax 文件(也叫做全局应用程序类文件)，是一个可选的文件，该文件包含响应 ASP.NET 或 HTTP 模块引发的应用程序级别事件的程序代码
.config	Web.config(Web 配置文件)，它向所在的目录和所有子目录提供配置信息
.resx	资源文件，是一个数据存储文件，是由应用程序部署的非可执行数据。在资源文件中存储数据，无需重新编译应用程序即可修改数据
.suo	解决方案用户选项，记录所有将与解决方案建立关联的选项，以便在每次打开时，它都包含用户所做的自定义设置
.pdb	PDB 是 Palm DataBase 的缩写，保存着调试和项目状态信息，从而可以对程序的调试配置进行增量链接
.xsd	XML Schema 是 DTD 的替代品。XML Schema 语言也就是 XML Schema Definition (XSD)
.vsdisco	项目发现文件，是基于 XML 的文件，它包含为 Web 服务提供发现信息的资源的链接(URL)
.htc	HTML 组件，全称就是 HTML Components，一个 HTML 文件，包含脚本和定义组件的一系列 HTC 特定元素。.htc 提供在脚本中实现组件的机制

续表

扩 展 名	说　明
.sitemap	站点地图文件，以 XML 格式定义存储的站点结构信息
.dll	动态链接库文件，全称是 Dynamic Link Library。所有 C#应用程序代码最终将编译成该文件，供 Web 窗体调用

3.2　ASP.NET指令

在基于面向对象思想的.NET 平台，可以称之为“万物皆对象”了。在这里，一个页面，一个用户控件，一个母版页等，全都是对象，全都有各自的属性。

在类文件里，我们表示类的属性可以直接声明。但是在页面文件里，我们就没有办法像类文件里那样声明属性，所以 ASP.NET 提供了指令机制，使我们能很方便快速地对页面属性进行设置，供分析器和编译器正确地编译和分析页面文件。

ASP.NET 指令有很多，这里收集了常见的 14 个，如表 3-2 所示。

表 3-2　ASP.NET指令表

指　令	说　明
@Application	定义 ASP.NET 应用程序编译器所使用的应用程序特定的属性，它只显示在应用程序文件(.asax)中
@Assembly	在编译期间将程序集链接到 ASP.NET 应用程序页，使程序集的所有类和接口都在该页上可用
@Control	定义用户控件的属性。包含在.ascx 文件(用户控件)中
@Implements	指定 Web 窗体或用户控件实现的接口
@Import	显式地导入命名空间
@Master	定义母版页的属性。包含在.master 文件(母版页)中
@MasterType	提供一种方法，当通过 Master 属性访问 ASP.NET 母版页时，创建对该母版页的强类型引用
@OutputCache	指定 Web 页面或用户控件使用的缓存策略
@Page	定义 Web 窗体使用的属性，这些属性将被 Web 窗体页分析器和编译器使用。包含在.aspx 文件中
@PreviousPageType	提供一种方法来获得上一页的强类型，可通过 PreviousPage 属性访问上一页
@Reference	将指定的窗体或用户控件链接到当前的窗体或用户控件
@Register	创建标记前缀和自定义控件之间的关联，这为开发人员提供了一种在 ASP.NET 应用程序文件中引用自定义控件的简明方法
@WebHandler	定义 HTTP 处理程序(.ashx)文件的特性和编译选项
@WebService	定义 ASP.NET 分析器和编译器使用的 XML Web Services 特定的(.asmx 文件)属性

ASP.NET 指令由一对尖括号和百分号括起来的前缀@符号和指令名称表示，指令里可以声明数个属性，例如：

```
<%@ Application Language="C#" %>
```

这是在全局应用程序文件 Global.asax 文件里定义的@Application 指令，该指令指明该文件内使用的程序语言为 C#。

ASP.NET 指令可以声明在文件的任意位置，不过为了规范，一般都位于文件的头部。

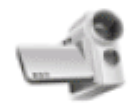
每一个指令都可以有一个或多个属性，属性和属性值成对出现。

3.2.1 提交合法的HTML标签

有时候我们需要让我们提交的文本展示出来的效果非常美观，通常会对服务器提交一些 HTML 标签来控制文本或内容的样式。

HTML 标签可能包含了很多不安全的因素，所以向服务器提交 HTML 标签通常会被服务器认为是不安全的操作。

ASP.NET 作为一个功能强大的应用系统，已经默认地把这类操作给过滤掉了。但是如果我们自己可以确保提交内容的安全性，当然可以通过设置，关闭该过滤选项。这就要用到 ASP.NET 页面的 Page 指令来对页面的安全性进行设置了。



视频教学： 光盘/videos/3/pageCommand.avi　　**长度：** 11 分钟

1. 基础知识——@Page指令

在 ASP.NET 里，使用最多的就是 Web 窗体。每一个 Web 窗体里都必须有@Page 指令，所以@Page 指令也是使用最为频繁的 ASP.NET 指令。

@Page 指令定义 Web 窗体使用的属性，这些属性将被 Web 窗体页分析器和编译器使用。只能包含在.aspx 文件中。

当我们每新建一个 Web 页面的时候，系统自动会为该 Web 页面头部创建一个@Page 指令，来指明页面最基本的属性。

初始代码为：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  Inherits="_Default" %>
```

其中<%@ Page [Attribute=Value] ... %>是系统默认的指令格式。后面由空格隔开的一组组的数据是该指令的属性和值。

这里的几个参数说明如下。

- Language="C#"：是指该页面的默认语言为 C#。
- AutoEventWireup="true"：是指系统自动绑定页面服务器端控件的事件。
- CodeFile="Default.aspx.cs"：是指定页面使用的代码文件。
- Inherits="_Default"：是说该页面继承自代码文件中的哪个类。

当然@Page 指令不只是有这几个属性。具体属性列表如表 3-3 所示。

表 3-3　@Page指令的属性

属　性	说　明
AutoEventWireup	用于指示页面上的服务器端控件的事件是否自动绑定
Buffer	用于确定是否启用了 HTTP 响应缓冲
ClassName	用于指定在请求页时将自动进行动态编译的页的类名，其值可以是任何有效的类名，并且可以包括类的完整命名空间；如果未指定该属性的值，则已编译页的类名将基于页的文件名
ClientTarget	指示 ASP.NET 服务器控件应该为其呈现内容的目标用户代理(通常是 Web 浏览器，该值可以是应用程序配置文件的<clientTarget>节中定义的任何有效别名
CodeFile	用于指定页引用的代码隐藏文件的路径。此属性与 Inherits 属性一起使用可以将代码隐藏源文件与网页相关联。该属性仅对编译的页有效
CodeFileBaseClass	指定页的基类及其关联的代码隐藏类的路径。该属性是可选的，如果使用该属性，必须同时使用 CodeFile 属性
CodePage	指示用于响应的编码方案的值，该值是一个用作编码方案 ID 的整数
ContentType	将响应的 HTTP 内容类型定义为标准的 MIME 类型
Debug	指示是否应使用调试符号编译该页
Description	提供该页的文本说明。ASP.NET 分析器忽略该值
EnableSessionState	定义页的会话状态要求。如果启用了会话状态，则为 true；如果可以读取会话状态但不能进行更改，则为 ReadOnly；否则为 false。默认值为 true。这些值是不区分大小写的
ErrorPage	定义出现未处理页异常时用于重定向的目标 URL
Inherits	定义供页继承的代码隐藏类。它可以是从 Page 类派生的任何类。它与 CodeFile 属性(包含指向代码隐藏类的源文件的路径)一起使用
Src	指定包含链接到页的代码的源文件的路径。在链接的源文件中，可以选择将页的编程逻辑包含在类中或代码声明块中

上面用到了一个 Inherits 属性，它用来设置页面与后台代码中相关联的类。我们打开 CodeFile 属性所指的文件，会找到该属性所指的类名。但是这里仅仅存放的是用户定义的事件处理程序，并没有任何服务器端对象的声明。

不过这里我们发现该类是一个 partial(部分)类，这说明系统还隐藏着一个文件，存放着该类的另一部分。而在这个隐藏的文件里有着所有服务器端控件的声明和属性设置代码。这两个部分类已经完整地声明了一个继承自 Page 类的类。

作为页面，因为其最终也要被系统解释成一个类，所以这里页面和后台代码类的关系是继承关系。所以，这里 Inherits 属性设置的是被继承的类。

2. 实例描述

作者闲得无聊，自己写了一个简单 HTML 编辑控件。

效果还不错，可就是用在 ASP.NET 程序中提交数据的时候老是说有潜在的危险。检查代

码并没有发现什么危险，只是提交的内容中有一些换行标签“
”。这是想让输入的内容在展示的时候换个行而已，却一直通不过。

很无奈！不过，查过资料才知道，在这个需要提交信息的页面的@Page 指令中简单地设置一下就好了。

这里我们为了更直观地展示效果，就直接在文本框中输入 HTML 文本进行提交，而不再使用 HTML 编辑控件了。

3. 实例应用

【例 3-1】我想提交合法的 HTML 标签。

(1) 我们先新建一个项目，新建一个 Web 窗体页面。

(2) 在前台页面拖入一个文本框，ID 属性设置为 txtContent。

(3) 在前台页面拖入一个命令按钮，ID 属性设置为 btnSubmit，Text 属性设置为 Submit。

(4) 换行，在页面中拖入一个标签控件，ID 属性设置为 lblShow。

(5) 在页面设计视图中双击命令按钮 btnSubmit，添加单击事件。

页面代码如下：

```
<asp:TextBox ID="txtContent" runat="server"></asp:TextBox>
<asp:Button ID="btnSubmit" runat="server" Text="Submit"
  onclick="btnSubmit_Click" />
<br />
<asp:Label ID="lblShow" runat="server" Text="Label"></asp:Label>
```

(6) 用户在单击命令按钮的时候，文本框里的值应当展示到标签里，这里我们修改命令按钮 btnSubmit 的事件处理程序，代码如下：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    this.lblShow.Text = this.txtContent.Text;
}
```

(7) 最后当然还要设置一下页面上的@Page 指令了，我们需要添加一个 ValidateRequest 属性，并设置其值为 false。修改后的@Page 指令代码如下：

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
  Inherits="_Default" ValidateRequest="false" %>
```

4. 运行结果

保存文件，运行项目，访问该页面。

我们先在文本框中输入“Hello!
你好！”字符串，然后单击 Submit 按钮，结果如图 3-1 所示。

可以看到在标签展示出来的结果中“Hello!”和“你好！”之间已经换了一行。

不过，至此我们并没有看到修改该属性的好处，我们删掉@Page 指令中对 ValidateRequest 属性的设置，然后再输入“Hello!
你好！”，单击 Submit 按钮，结果如图 3-2 所示。

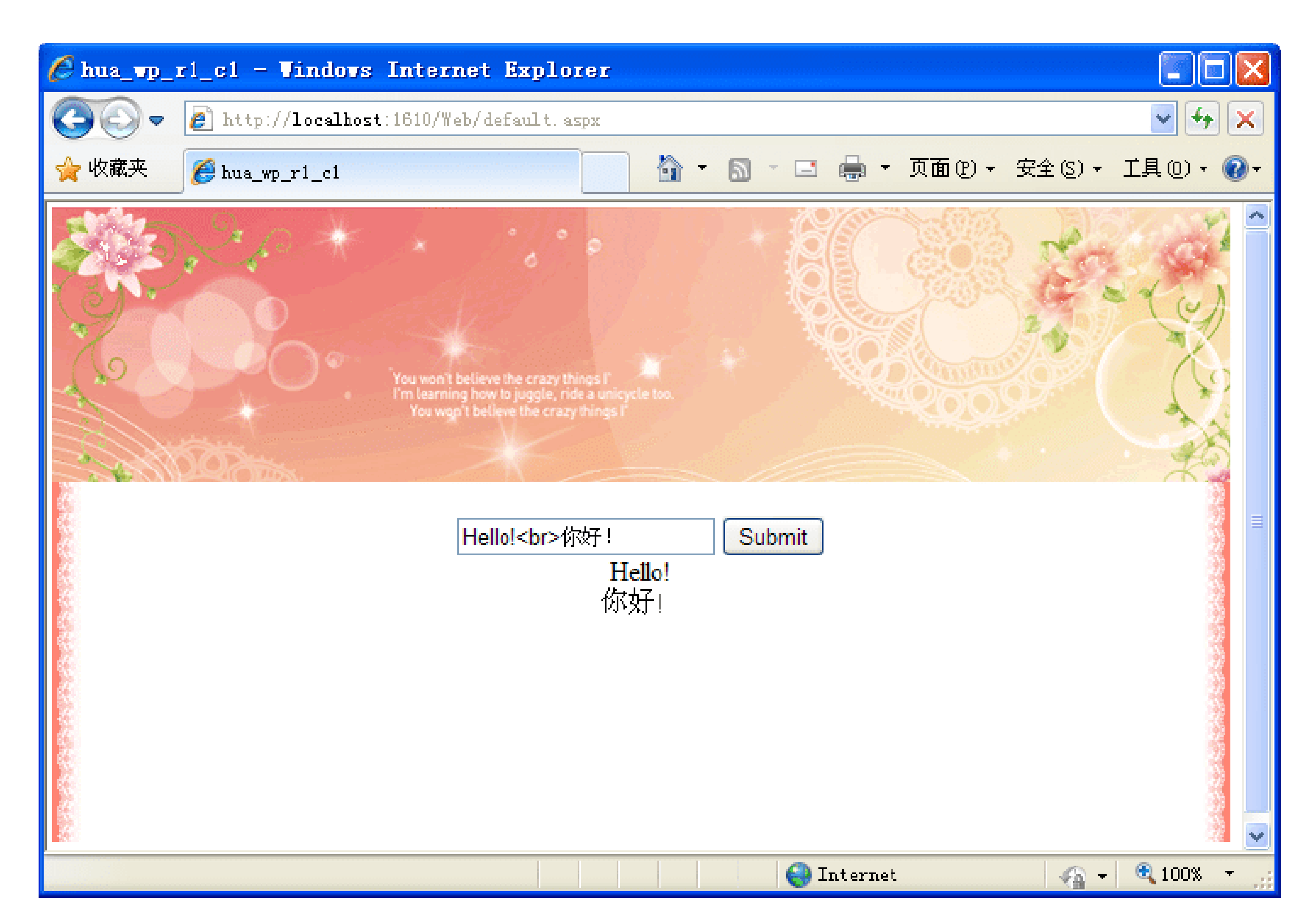


图 3-1　运行结果



图 3-2　删除ValidateRequest属性后的运行结果

可以看到，出错了。检测到有潜在危险的 Request.Form 值。

5. 实例分析



源码解析：

本实例创建一个 TextBox 控件、一个 Button 控件和一个 Label 控件，在单击 Button 控件提交表单的时候，将 TextBox 控件里的值使用 Label 控件展示出来。

在服务器提交过程中，系统自动验证请求的安全性，如果检查到有 HTML 标签，则系统会认为该次请求是不安全的。我们需要关闭本页面对请求数据的验证，设置本页面@Page 指令的 ValidateRequest 属性值为 false。

3.2.2 使用用户控件

ASP.NET 提供了用户自定义控件机制来封装某些独立的功能，或者是页面的某个模块。通过这些封装，使页面代码更加结构化，也提高了页面代码的复用性。

用户自定义控件可以包含任何 HTML 代码和服务器端控件，所以它需要具有类似于页面一样的可随意编辑的特性。但是它毕竟不是页面，所以不能被当做页面处理。所以，它有自己的文件格式、自己的属性、自己独有的页面指令。

 **视频教学：光盘/videos/3/ControlCommand.avi　　长度：11 分钟**

1. 基础知识——@Control指令和@Register指令

ASP.NET 用户控件的页面指令是@Control。与 Web 窗体的@Page 指令用法一样，用来定义用户控件的属性，供分析器的编辑器检查使用。

当我们新建一个用户控件的时候，系统会自动地在该用户控件的头部加入一个@Control 指令。默认代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true"
  CodeFile="WebUserControl.ascx.cs" Inherits="WebUserControl" %>
```

这里的属性和@Page 指令一样，Language 指定服务器端语言；AutoEventWirup 说明是否自动绑定事件；CodeFile 指定用户控件后台代码文件；Inherits 同样是指定该用户控件继承自哪一个类。

下面是具体的属性列表，如表 3-4 所示。

表 3-4　@Control指令属性

属　性	说　明
AutoEventWireup	用于指示控件上的服务器端控件的事件是否自动绑定
ClassName	一个字符串，用于指定需在请求时进行动态编译的控件的类名。此值可以是任何有效的类名，并且可以包括类的完整命名空间(一个完全限定的类名)。如果没有为此属性指定值，已编译控件的类名将基于该控件的文件名
CodeFile	指定所引用的控件代码隐藏文件的路径。此属性与 Inherits 属性一起使用，将代码隐藏源文件与用户控件相关联
Debug	指示是否应使用调试符号编译控件
EnableViewState	指示是否跨控件请求维护视图状态。如果维护视图状态，则为 true；否则为 false。默认值为 true
Inherits	定义提供控件继承的代码隐藏类。它可以是从 UserControl 类派生的任何类。与包含代码隐藏类源文件的路径的 CodeFile 属性一起使用
Src	指定包含链接到控件的代码的源文件的路径。在所链接的源文件中，您可选择在类中或在代码声明块中包括控件的编程逻辑

在声明用户控件的时候，我们要用到@Control 指令，在使用用户控件的时候，我们需要用到@Register 指令。@Register 指令的语法为：

```
<%@ Register src="WebUserControl.ascx" tagname="WebUserControl"
  tagprefix="uc1" %>
```

其具体属性说明如表 3-5 所示。

表 3-5　@Register指令的属性

属　性	说　明
tagprefix	一个任意别名，它提供对包含指令的文件中所使用的标记的命名空间的短引用
tagname	与类关联的任意别名。此属性只用于用户控件
namespace	正在注册的自定义控件的命名空间
src	与 tagprefix:tagname 对关联的声明性 ASP.NET 用户控件文件的位置(相对的或绝对的)
assembly	与 tagprefix 属性关联的命名空间所驻留的程序集

用户控件可以使用在 Web 窗体或其他用户控件中，但是绝对不能自己嵌套自己。所以@Register 指令可以出现在用户控件或 Web 窗体内。

2. 实例描述

在浏览一些大的门户网站的时候，我们经常会见到一个出现在不同页面不同位置的搜索框。这些搜索框一般由一个文本框、一个按钮和一些其他的东西组成。虽然在不同的位置，却是同一个风格，同一个模样。

作者刚好这段时间要做一个网站，网站里有站内搜索的功能，用的是用户控件，感觉相当不错。使用后不仅代码结构清晰了，而且修改的时候更方便了。

3. 实例应用

【例 3-2】使用用户控件。

(1) 打开上节我们使用的项目，添加一个“Web 用户控件”项，命名为 WUCSearch.ascs。

(2) 添加控件内容。这里添加一个文本框和一个命令按钮。添加后用户控件内的代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="WUCSearch.ascx.cs"
  Inherits="WUCSearch" %>
<h2>Search</h2>
<input type="text" name="key" id="searchtext" />
<input type="submit" id="searchsubmit" value="search" />
```

(3) 打开需要添加搜索框的页面，这里是 Index.aspx。

(4) 在页面代码头部的@Page 指令下面添加一行代码，如下所示：

```
<%@ Register src="WUCSearch.ascx" tagname="WUCSearch" tagprefix="uc1" %>
```

该代码将把刚才我们创建的用户控件 WUCSearch.ascx 注册到这个页面上。以后就可以直接在这个页面的任何地方使用了。

(5) 在要添加搜索框的地方加入如下一行代码：

```
<ucl:WUCSearch ID="wucSearchControl" runat="server" />
```

该代码将引入用户控件。

切换到“设计”视图，效果如图 3-3 所示。

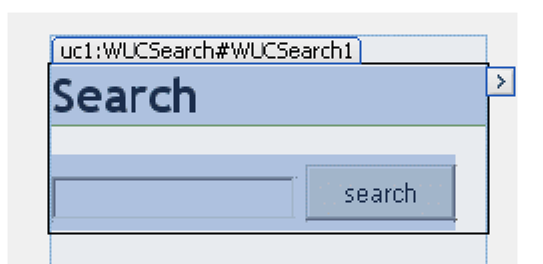


图 3-3　搜索框的“设计”视图效果

4. 运行结果

保存所有文件，运行项目。

访问 Index.aspx 页面，结果如图 3-4 所示。

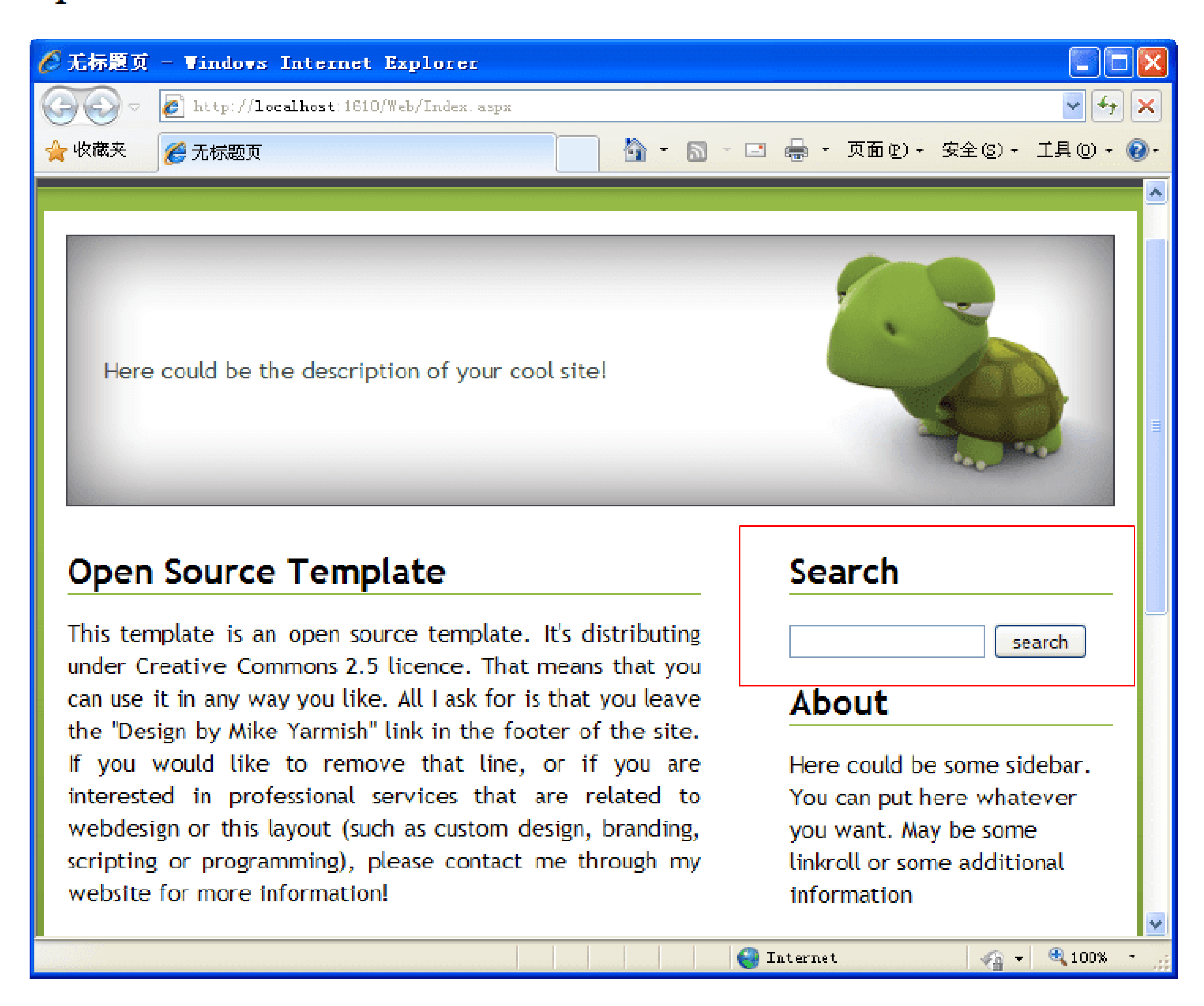


图 3-4　运行结果

图 3-4 中，红框内的部分即为使用用户控件执行的效果。

5. 实例分析



源码解析：

实例添加了一个“Web 用户控件”项，然后添加该用户控件的内容。添加用户控件的时候系统自动添加了@Control 指令。接下来在要使用该用户控件的页面注册该用户控件，在要加入该用户控件的地方直接像一般服务器控件一样使用即可。

其实在使用用户控件的时候完全可以把“解决方案资源管理器”窗口里的用户控件直接拖动到相应页面的“设计”视图中的对应地方即可。

3.2.3　缓存整个页面

在一些大型门户系统中，访问量特别大，可能每分钟就会有上千次的访问。而且通常门户系统的首页非常复杂，项目特别多，每一次访问都需要多次操作数据库。这对服务器的性能是个很大的考验。

但是我们发现通常服务器首页更新的频率不是非常快。所以，完全可以让服务器的首页缓存起来，过一段时间再更新。这样服务器就不用每次请求都重新读取数据库并生成页面了。

刚好，ASP.NET 为我们提供了一个@OutputCache 指令，我们只要在需要缓存数据的页面使用该指令，即可实现对整个页面进行缓存。



视频教学：光盘/videos/3/OutputCacheCommand.avi　　长度：6 分钟

1. 基础知识——@OutputCache指令

@OutputCache 指令用于以声明的方式控制 ASP.NET 页或页中包含的用户控件的输出缓存策略。格式如下：

```
<%@ OutputCache Duration="5" VaryByControl="none" %>
```

这行语句的意思是无条件地为该页缓存 5 秒钟。

@OutputCache 指令的属性如表 3-6 所示。

表 3-6　@OutputCache指令属性

属　性	说　明
Duration	页或用户控件进行缓存的时间(以秒计)
Location	用于指定输出缓存项的位置，OutputCacheLocation 枚举值之一。默认值为 Any
Shared	一个布尔值，确定用户控件输出是否可以由多个页共享。默认值为 false
VaryByCustom	表示自定义输出缓存要求的任意文本
VaryByHeader	分号分隔的 HTTP 标头列表，用于使输出缓存发生变化。将该属性设为多标头时，对于每个指定标头组合，输出缓存都包含一个不同版本的请求文档
VaryByParam	分号分隔的字符串列表，用于使输出缓存发生变化
VaryByControl	一个分号分隔的字符串列表，用于更改用户控件的输出缓存
CacheProfile	用于定义与该页关联的缓存设置的名称



在@OutputCache 指令的所有属性中，其中 Duration 和 VaryByParam 属性是必选的属性。

2. 实例描述

为了方便，这里我们就不再模拟一个大型门户系统了。

我们只在页面上打印当前时间，并让页面每秒钟刷新一次。这样的话，每一次刷新打印的时间都会不一样。

在这个前提下我们将该页面缓存 5 秒，如果缓存成功，则 5 秒内页面数据不会变化，在第 6 秒时才更改为新的时间值。

3. 实例应用

【**例 3-3**】缓存整个页面。

(1) 打开上节使用的项目，创建一个新的 Web 窗体，命名为 OutputCache.aspx。

(2) 编辑该 Web 窗体，在页面里添加如下两行代码：

```
<%=DateTime.Now.ToString() %>
<script>setTimeout("location.href=location.href",1000)</script>
```

第一行在页面中打印当前时间，第二行使页面每隔 1 秒钟刷新一次。

(3) 在页面头部添加缓存代码如下：

```
<%@ OutputCache Duration="5" VaryByControl="none" %>
```

4. 运行结果

保存所有文件，运行项目。

访问 OutputCache.aspx 页面，结果如图 3-5 所示。

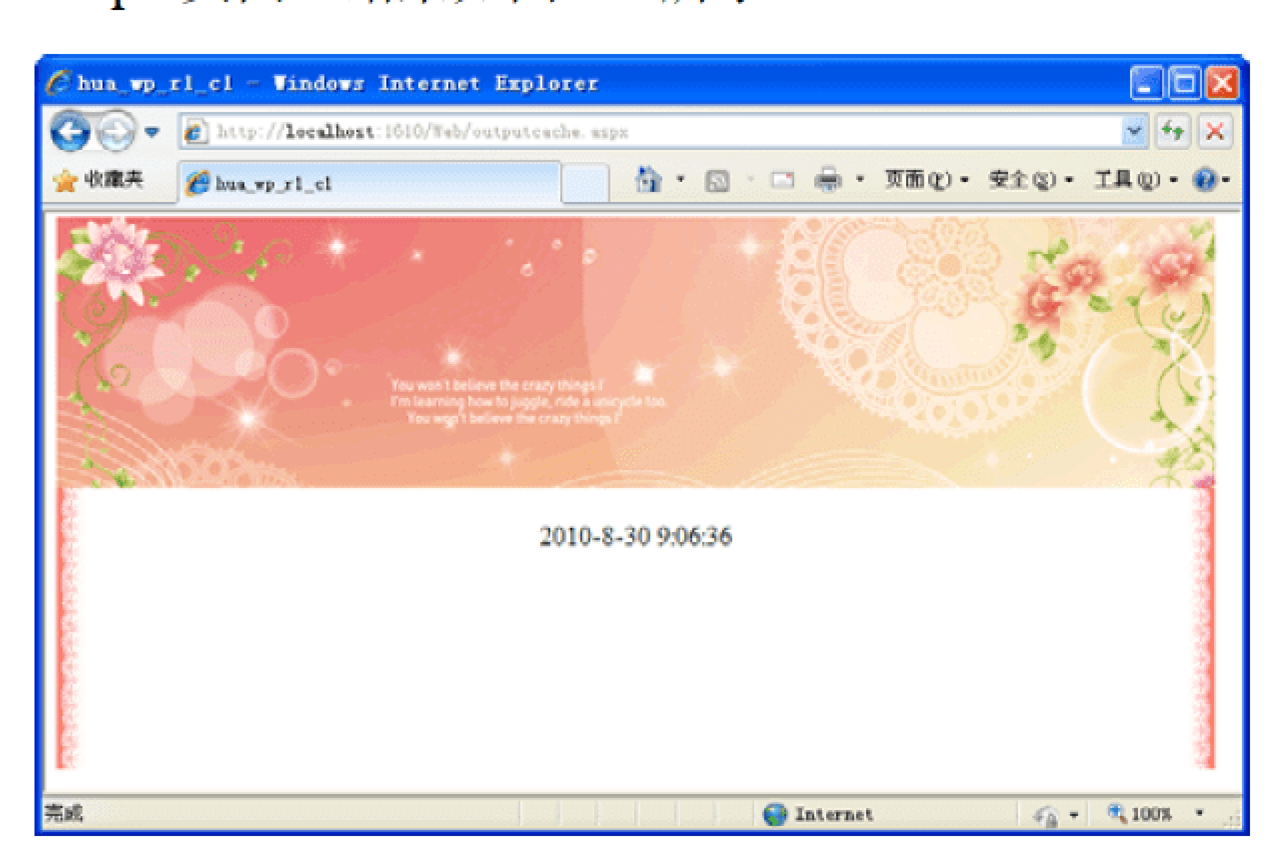


图 3-5　OutputCache.aspx页面

该页面将每秒钟更新一次，但是页面内容会缓存 5 次，在第 6 次访问的时候更新。

5. 实例分析


源码解析：

该实例非常简单，在页面打印当前时间，并让页面每秒钟刷新一次。然后设置页面缓存 5 秒钟。

其实细心的读者可能会发现页面时间每次更新都会跨越 6 秒，而不是我们想象中的 5 秒钟换一个值。这是因为系统计算时间是在第一次访问开始，到第 5 次刷新时刚好 5 秒，所以这次显示的还是第一次访问的时间。处理完该次请求，缓存的数据作废，在第 6 次刷新(即第 7 次访问)的时候页面才重新生成并缓存页面。

3.3　获取请求信息

在 B/S 结构的应用程序中，客户端要向服务器端发出请求，请求的信息包括客户端信息、请求的 URL、请求的参数和 Cookie 等内容。

服务器在接收到用户请求时自动将请求封装到 Request 对象中供我们使用。所以 Request 对象对于我们来说是一个非常重要的对象。

Request 对象是一个 System.Web.HttpRequest 类的对象。系统将其作为一个只读的公有属性内置在 Page 类中。我们可以在每一个 Page 类的对象中直接使用。Request 对象提供了一系列的属性和方法来完成上述功能。常用的方法和属性如表 3-7 所示。

表 3-7　Request对象的方法和属性

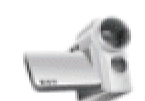
名　称	说　明
BinaryRead 方法	执行对当前输入流进行指定字节数的二进制读取
MapImageCoordinates 方法	将传入图像字段窗体参数映射为适当的 x 坐标值和 y 坐标值
MapPath 方法	为当前请求将请求的 URL 中的虚拟路径映射到服务器上的物理路径
SaveAs 方法	将 HTTP 请求保存到磁盘
ValidateInput 属性	对通过 Cookies、Form 和 QueryString 属性访问的集合进行验证
Browser 属性	获取或者设置有关正在请求的客户端的浏览器功能的信息
ContentLength 属性	指定客户端发送的内容长度
Cookies 属性	获取客户端发送的 Cookie 的集合
Form 属性	获取窗体变量集合
QueryString 属性	获取 HTTP 查询字符串变量集合
IsLocal 属性	获取一个值，该值指示该请求是否来自本地计算机
RawUrl 属性	获取当前请求的原始 URL
Url 属性	获取有关当前请求的 URL 的信息
UserHostAddress 属性	获取远程客户端的 IP 主机地址
UserHostName 属性	获取远程客户端的 DNS 名称

3.3.1　获取客户端信息

两个人要想友好地交往，肯定得互相了解。

但浏览器做一次请求，仓促地握一次手很难会有什么印象。所以，浏览器在每一次请求的时候都会把自我介绍的信息一并封装提交给服务器，向服务器做一个完整的自我介绍。服务器接收过以后会把用户请求的所有信息封装到 Request 对象中。

本小节我们就使用 Request 对象来查看一下客户端浏览器的主机名、IP 地址以及当前请求的 URL。



视频教学：光盘/videos/3/RequestObject.avi　　长度：6 分钟

1. 基础知识——Request对象

Request 对象提供了一系列的属性来获取封装后的客户端请求信息。

其中属性 UserHostName 和属性 UserHostAddress 二者分别用来获取客户端浏览器所在的主机名和主机 IP。

属性 UserHostName 和属性 UserHostAddress 的返回值都是 string 类型。UserHostName 返回的是远程客户端的 DNS 名称，假如客户端在 DNS 中心没有注册，则返回客户端的 IP 地址。

Request 对象的 Url 属性用来获取有关本次请求的 URL 的信息。Url 属性是 System.Uri 类型的对象，该对象提供统一资源标识符(URI)的对象表示形式和对 URI 各部分的轻松访问。

System.Uri 类提供了诸多属性供我们使用。其中有获取主机名的 Host 属性，获取请求端口的 Port 属性等。

如果只想获取地址栏 URL 的全部信息，直接调用 ToString()方法即可。

2. 实例描述

在很多门户网站中，经常会根据各位访客所处的不同地理位置而有选择地推荐一些新闻资料供用户查看。

而确定访客地理位置的唯一办法就是 IP 地址，服务器通常获取每位访客的 IP 地址，根据 IP 地址库确定访客的位置，进行有选择的查询并给用户展示一些新闻信息。

本实例我们就结合刚才讲过的知识，获取访客的 IP 地址并展示给访客。

3. 实例应用

【例 3-4】获取客户端信息。

(1) 运行上一节我们使用的项目。

(2) 创建一个 Web 窗体，命名为 BroswerInfo.aspx。

(3) 在页面中拖入 3 个 Label 控件，分别命名为 lblHostName、lblIP 和 lblUrl。

代码如下：

```
主机名：<asp:Label ID="lblHostName" runat="server"></asp:Label><br />
IP 地址：<asp:Label ID="lblIP" runat="server"></asp:Label><br />
URL：<asp:Label ID="lblUrl" runat="server"></asp:Label><br />
```

(4) 修改后台代码，为页面上的 3 个标签赋值。

将 Page_Load 方法修改为如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lblHostName.Text = Request.UserHostName;    //Host Name
    this.lblIP.Text = Request.UserHostAddress;    //IP
    this.lblUrl.Text = Request.Url.ToString();    //URL
}
```

(5) 保存所有文件。

4. 运行结果

运行项目，访问 BroswerInfo.aspx 页面。结果如图 3-6 所示。

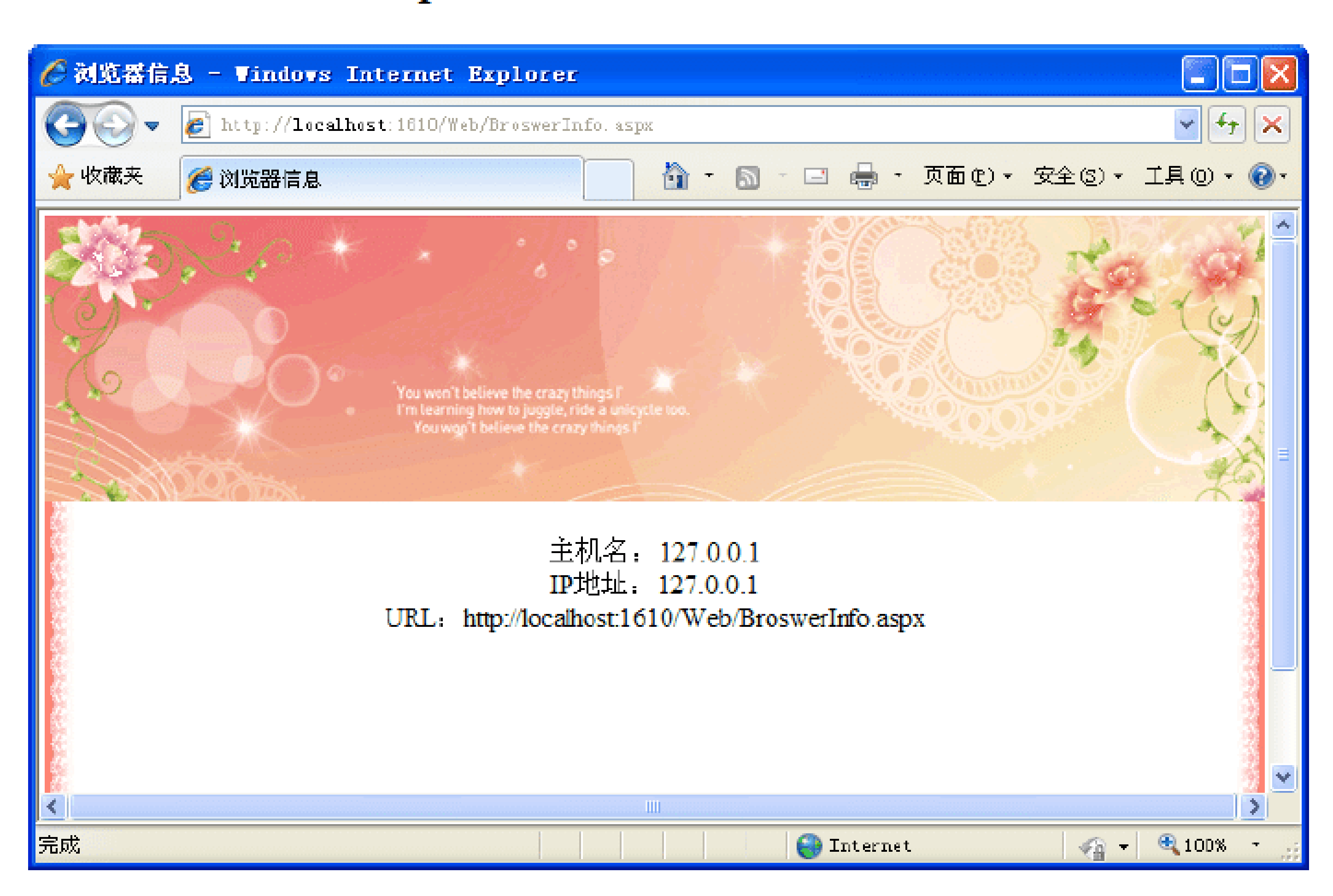


图 3-6　查看客户端信息

5. 实例分析



源码解析：

本实例使用 Request 对象的 UserHostName 属性、UserHostAddress 属性和 Url 属性分别获取客户端浏览器所在主机的 DNS 名称、IP 地址和 URL 信息，并将其展示到被请求的页面上。

3.3.2　遍历当前浏览器头信息

在使用 B/S 结构的项目时，我们会觉得非常简单，只需要在地址栏输入想要访问的 URL，即可得到想要的信息。实际上在这个过程中浏览器和服务器在这一个请求中做了许多工作。

浏览器收集本机信息和请求封装成一个数据包发送到服务器。在这个数据包的头部封装了客户端浏览器的信息，比如客户端语言、浏览器版本等内容。在服务器端可以接收并查看这些信息，或以此做相应的操作。

 **视频教学：**光盘/videos/3/UseRequestObject.avi　　　　长度：6 分钟

1. 基础知识——Header属性

Request 对象可以使用 Header 属性来访问封装过的浏览器头信息。

Request 对象的 Header 属性是一个 System.Collections.Specialized.NameValueCollection 类型的对象。该类提供了许多方法，以执行对浏览器头信息进行读取、添加或者删除的操作。

Header 属性是一个集合，以"键-值"对的方式存储了浏览器请求的所有信息。使用其 Keys 属性能获取该对象的所有键的集合。我们可以使用该集合的 Count 属性得到其长度，然后遍历该集合，取出所有键，再用键名从 Header 中取出所有的浏览器头信息。

2. 实例描述

了解过了 Web 请求的数据包后，出于好奇，我们来遍历查看一下浏览器头信息的内容。

3. 实例应用

【例 3-5】遍历当前浏览器的头信息。

(1) 打开上次使用的项目。

(2) 新建一个 Web 窗体，命名为 HeaderInfo.aspx。

(3) 向页面中拖入一个 Label 标签，命名为 lblHeader，做展示浏览器头信息之用：

```
<asp:Label ID="lblHeader" runat="server"></asp:Label>
```

(4) 修改后台代码 Page_Load 方法，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    int count = Request.Headers.Keys.Count;    //键集合的大小
    for (int i=0; i<count; i++)
    {
        string key = Request.Headers.Keys[i];    //取出一个键
        /* 添加内容到页面标签控件中 */
        this.lblHeader.Text += key + " : ";
        this.lblHeader.Text += Request.Headers[key];
        this.lblHeader.Text += "<br>";
    }
}
```

(5) 保存，完成实例。

4. 运行结果

运行项目，使用 IE 浏览器访问 HeaderInfo.aspx 页面，结果如图 3-7 所示。

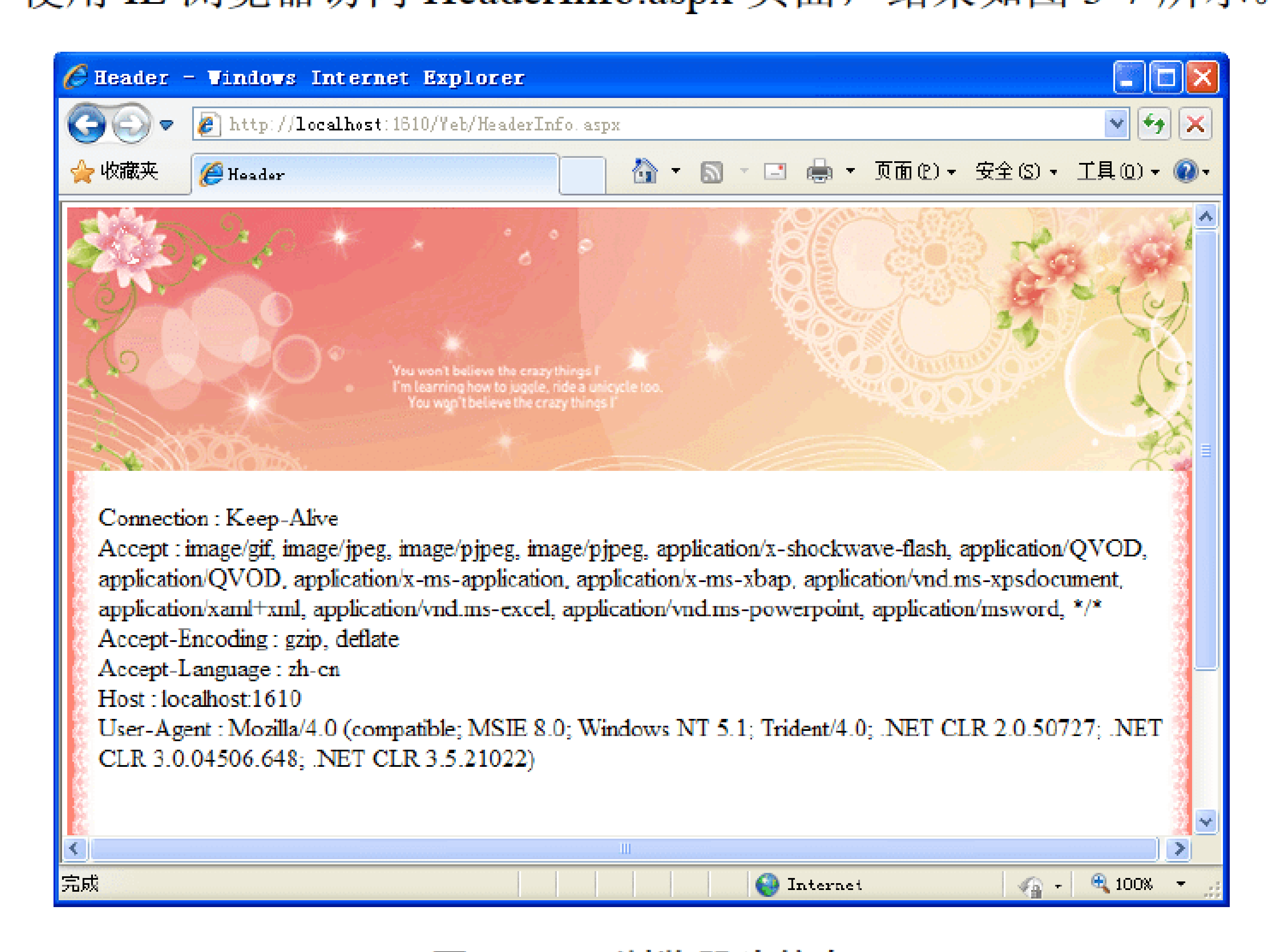


图 3-7 IE浏览器头信息

既然是浏览器头信息，那么对于不同的浏览器，信息的内容应该不尽相同吧。我们用 Chrome 浏览器访问该页面，结果如图 3-8 所示。

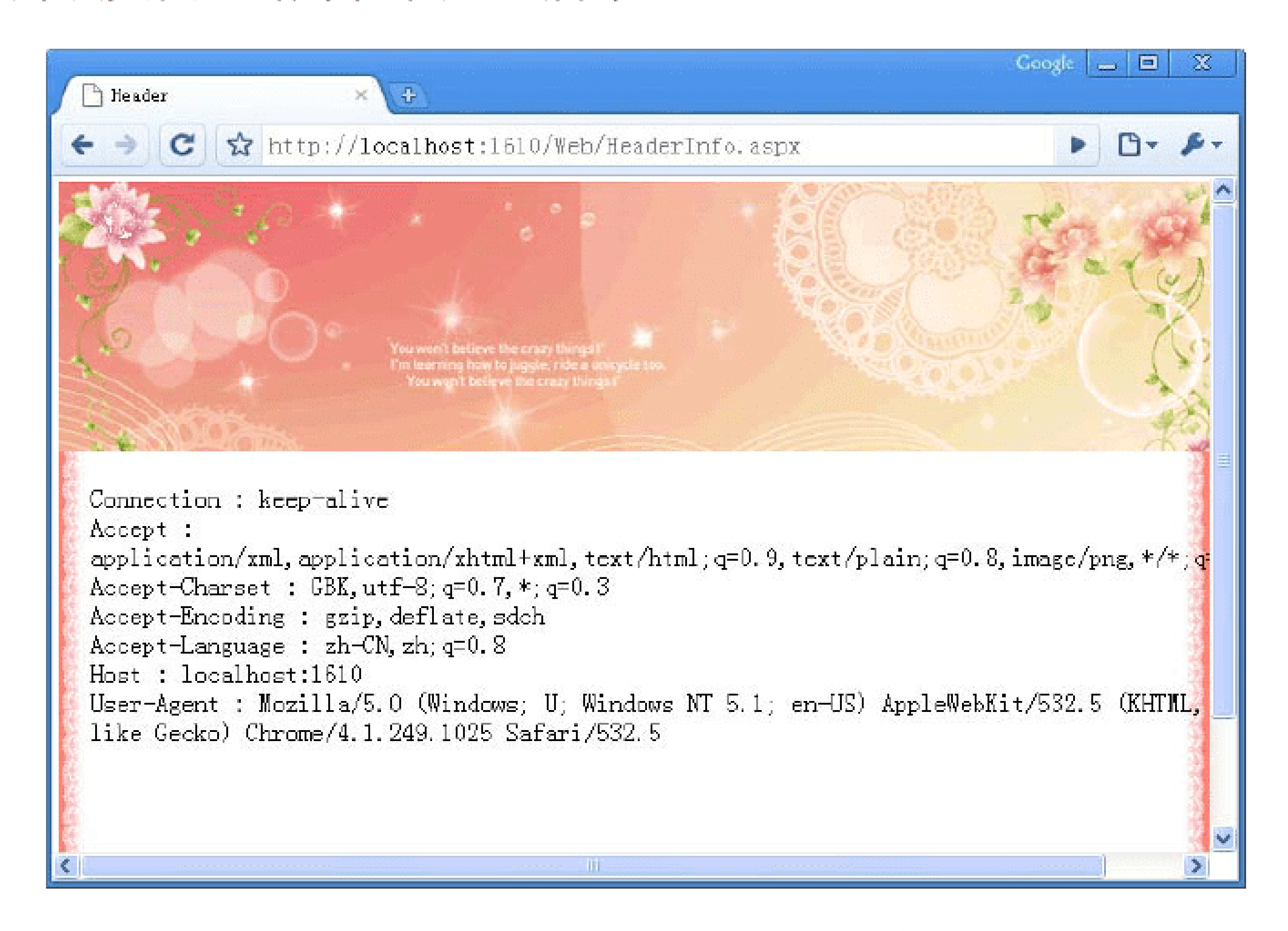


图 3-8　Chrome浏览器头信息

5. 实例分析



源码解析：

实例遍历了 Request 对象的 Header 属性的 Keys 集合，取出键以后以键名获得对应键的值，并向页面展示了浏览器头信息的所有的键和对应的值。

因为浏览器不同，所以同样的访问封装的数据也不尽相同。从返回信息中可以知道浏览器能接收的文档类型、字符集、编码、语言、请求主机地址和浏览器版本及环境等详细信息。

3.3.3　接收提交数据

互联网自诞生那天起，所涵盖的数据量就开始以几何数字增长。就眼下而言，数据已经多得难以计量。我们如何对它们管理呢？

当然，现在使用最多的就是 Web 应用程序。

Web 程序与客户端浏览器身居两地，在执行数据增删改查操作的时候通常要向 Web 服务器提交一些数据，服务器端根据请求的数据执行相应的操作。

那么在 ASP.NET 中我们如何获取客户端提交的数据呢？

其实在上一节我们已经提到过了，是 Request。



视频教学：光盘/videos/3/UseRequestObject.avi　　　　长度：6 分钟

1. 基础知识——Form属性和QueryString属性

客户端请求最常用的有两种方式：POST 和 GET。

POST 方式将数据打包隐式发送，GET 方式直接把数据追加到 URL 的后面发送给服务器。二者各有优劣，这里不再详述。

在 ASP.NET 的 Request 对象中提供了 Form 和 QueryString 两个属性，分别用来获取 POST 和 GET 两种提交方式提交过来的信息。

这两种方式都可以使用键名或索引查询。

格式如下：

```
Request.Form["key"];
Request.Form[0];
Request.QueryString["key"];
Request.QueryString[0];
```

另外，还有一种综合的查询方式：Request 对象的索引器。使用 Request 的索引器可以查询包括 Cookies、Form、QueryString 或者 ServerVariables 里的值。

不过，这种方式的话，没办法建立索引，就只能以键名来查询参数了。

格式如下：

```
Request["key"];
```

2. 实例描述

接收提交请求数据这个功能，任何 Web 应用程序都肯定会用到，就不用举特例了。这里简单地写一个页面，看一下各种获取提交值的方法。

3. 实例应用

【例 3-6】接收提交数据。

(1) 打开上次我们使用的项目。

(2) 创建一个新的 Web 窗体，命名为 SubmitValue.aspx。

(3) 修改页面 Body 部分，代码如下：

```
<form action="SubmitValue.aspx?key1=ThisIsKey1(GET)&key2=ThisIsKey2(GET)"
  method="post">
Key2: <input name="key2" /><br />
Key3: <input name="key3" /><br />
<input type="submit" value="Submit" />
</form>
<asp:Label ID="lblValue" runat="server"></asp:Label>
```

(4) 在后台，修改 Page_Load 方法，添加代码，如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.lblValue.Text = "";
    this.lblValue.Text += "<br>Form Key1=" + Request.Form["key1"];
    this.lblValue.Text += "<br>Form Key2=" + Request.Form["key2"];
    this.lblValue.Text += "<br>Form Key3=" + Request.Form["key3"];
```

```
    this.lblValue.Text += "<br>QueryString Key1=" + Request.QueryString["key1"];
    this.lblValue.Text += "<br>QueryString Key2=" + Request.QueryString["key2"];
    this.lblValue.Text += "<br>QueryString Key3=" + Request.QueryString["key3"];
    this.lblValue.Text += "<br>Key1=" + Request["key1"];
    this.lblValue.Text += "<br>Key2=" + Request["key2"];
    this.lblValue.Text += "<br>Key3=" + Request["key3"];
}
```

这里我们使用了 3 种方式各获取一次提交过去的 3 个键值，展示到页面上。

(5)　最后保存。

4. 运行结果

运行项目，访问 SubmitValue.aspx 页面。

在 Key2 文本框中输入“ThisIsKey2(POST)”，在 Key3 文本框中输入“ThisIsKey3(Post)”，然后单击 Submit 按钮，结果如图 3-9 所示。

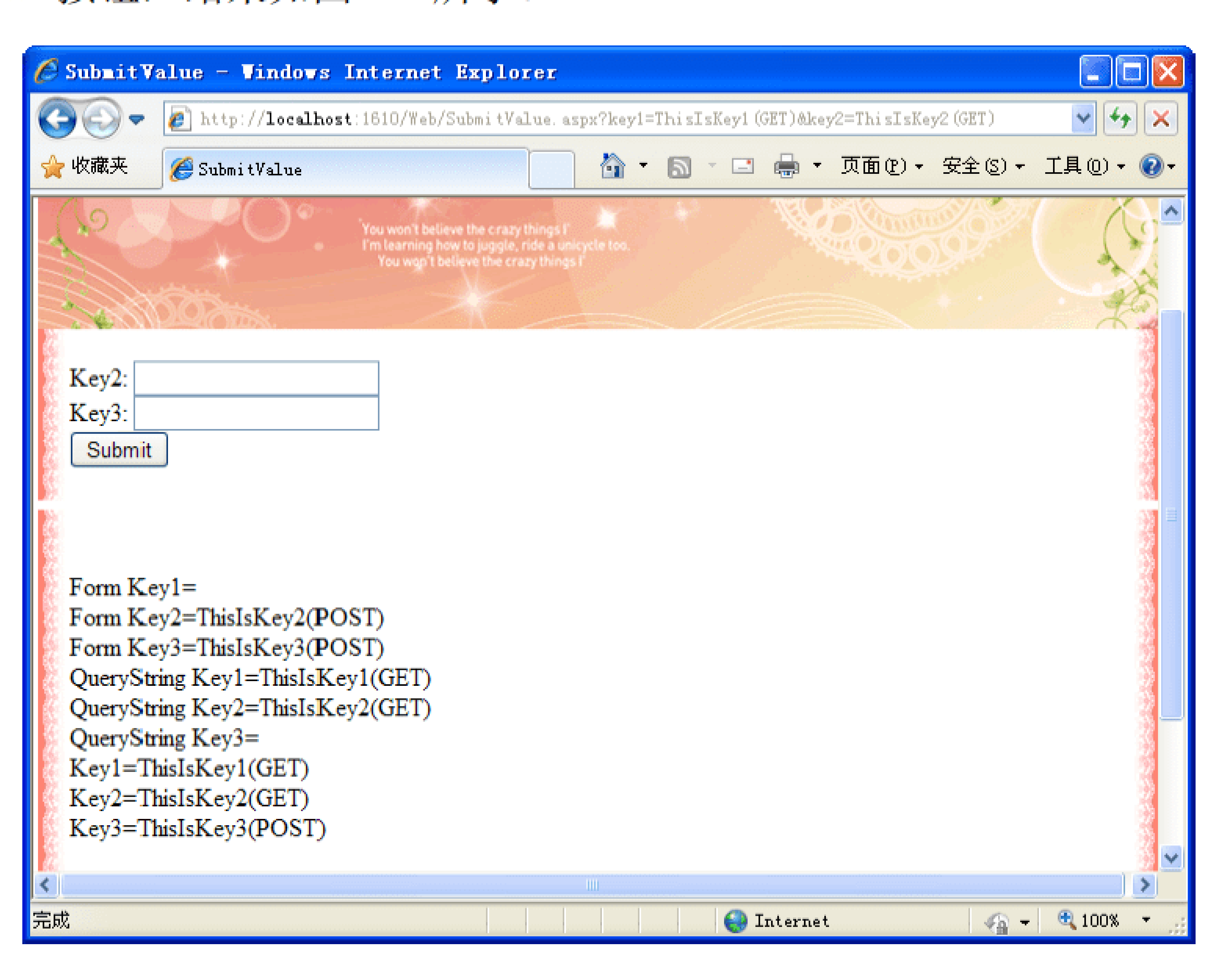


图 3-9　Request取值

5. 实例分析



源码解析：

实例使用 POST 和 GET 两种方式提交数据，然后使用 Form 属性、QueryString 属性和 Request 索引器三种方式分别取值。

在图 3-9 中我们可以看到，对表单里以 POST 方式提交的数据 QueryString 访问不到，对 URL 里以 GET 方式提交的数据 Form 访问不到，而使用 Request 索引器可以访问到任何方式提交的数据。不过当这两种方式有冲突时，首先采用 GET 方式传递的值。

3.4 向客户端输出信息

上一节我们研究了客户端的请求。有请求就得有响应。我们获得了客户端请求的数据，处理完以后我们就要对用户请求做出响应了。

沿袭传统，ASP.NET 里还使用 Response 对象对用户的请求做出响应。

Response 对象用于动态响应用户请求，控制发送给用户的信息，包括向页面打印文本、控制页面转向、创建图像、发送 Cookie 等功能。

Response 对象是 System.Web.HttpResponse 类的对象。系统将其作为一个只读的公有属性内置在 Page 类中，我们可以在任何一个 Page 类中直接使用。

Response 对象的常用方法和属性如表 3-8 所示。

表 3-8　Response对象方法和属性

名　称	说　明
AppendCookie 方法	基础结构。将一个 HTTPCookie 添加到内部 Cookie 集合
AppendHeader 方法	将 HTTP 头添加到输出流
BinaryWrite 方法	将一个二进制字符串写入 HTTP 输出流
Clear 方法	清除缓冲区流中的所有内容输出
Close 方法	关闭到客户端的套接字连接
End 方法	将当前所有缓冲的输出发送到客户端，停止该页的执行，并引发 EndRequest 事件
Flush 方法	向客户端发送当前所有缓冲的输出
Redirect 方法	将客户端重定向到新的 URL
SetCookie 方法	基础结构。更新 Cookie 集合中的一个现有 Cookie
Write 方法	将信息写入 HTTP 响应输出流
WriteFile 方法	将指定的文件直接写入 HTTP 响应输出流，即读取文件并写入客户端输出流
WriteSubstitution 方法	允许将响应替换块插入响应，从而允许为缓存的输出响应动态生成指定的响应区域
BufferOutput 属性	获取或设置一个值，该值指示是否缓冲输出并在处理完整个页之后发送
Charset 属性	获取或设置响应数据流的字符集
ContentType 属性	获取或设置输出流的 HTTP MIME 类型。默认值是 text/html


视频教学： 光盘/videos/3/UseResponseObject.avi　　长度：6 分钟

3.4.1 输出HTML文本

在后台程序中经常需要向页面输出一些 HTML 文本。例如有时为了在客户端弹出一个对话框，经常需要向页面打印一个包含 alt 方法的 Script 标签，好让客户端浏览器在访问该页面的时候执行一些提示信息。

Response 对象的 Write 方法就能向页面打印一些文本信息，我们可以将 Script 标签的内容作为一行文本打印到页面上。

1. 基础知识——Write方法

Response 对象有一个很重要的功能，就是将文本信息显示在客户端浏览器上，该功能使用 Write 方法实现。该方法可以将字符、对象、字符串、字符数组等信息显示在客户端浏览器上。

该方法有 4 个重载，分别用于显示上述 4 类信息，如下所示。

- Write(char ch)：将一个字符写入 HTTP 响应流。
- Write(object obj)：将一个对象写入 HTTP 响应流。
- Write(string s)：将一个字符串写入 HTTP 响应流。
- Write(char []buffer, int index, int count)：将一个字符数组写入 HTTP 响应流。后面两个参数一个是指定数组的起始位置，一个是指定写入响应流的字符数组的长度。

2. 实例描述

在实现系统登录功能的时候，经常会验证并提示用户登录操作的执行结果。

拿我们公司这个内部的文档管理系统来说，在登录的时候要求假如用户不存在，则弹出提示框告知用户。

实现的时候就使用了 Response 对象的 Write 方法向页面打印一组 Script 标签，弹出一个提示框来提示用户操作执行结果。

3. 实例应用

【例 3-7】输出 HTML 文本。

(1) 新建一个 Web 项目。

(2) 管理系统的默认页面是登录页面，所以我们这里在 Default.aspx 页面直接修改。

(3) 登录功能需要一个“用户名”文本框，一个密码框，一个“登录”命令按钮。

(4) 在“登录”命令按钮单击时，我们需要验证用户名是否存在。所以我们修改“登录”按钮的单击事件处理程序。

代码如下：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (this.txtUsername.Text != "admin")
    {
        this.Response.Write("<script>alert('用户名不存在！');</script>");
    }
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面。

输入用户名和密码，如图 3-10 所示。

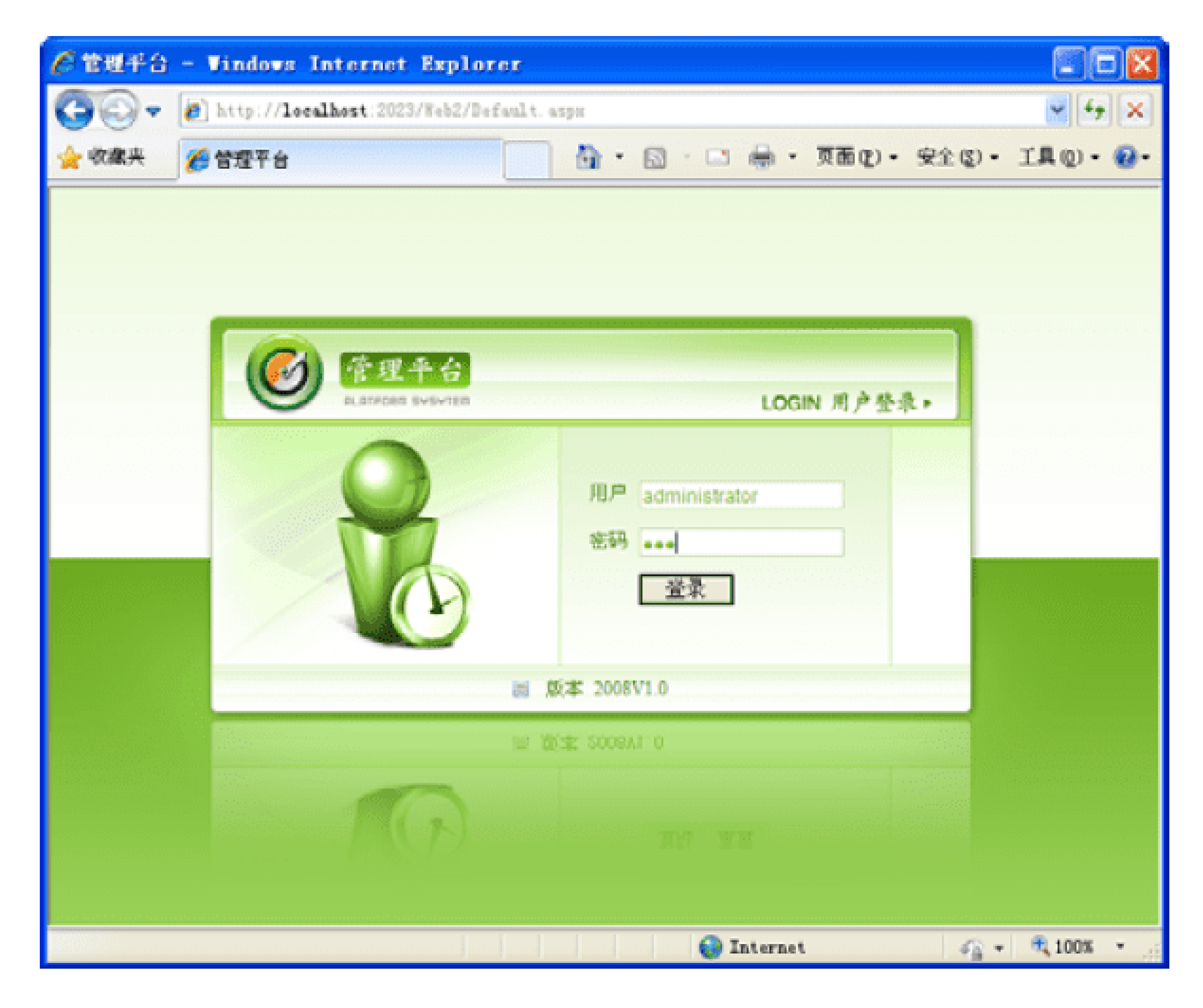


图 3-10　管理系统登录

单击“登录”按钮，系统执行单击事件，弹出对话框提示，如图 3-11 所示。

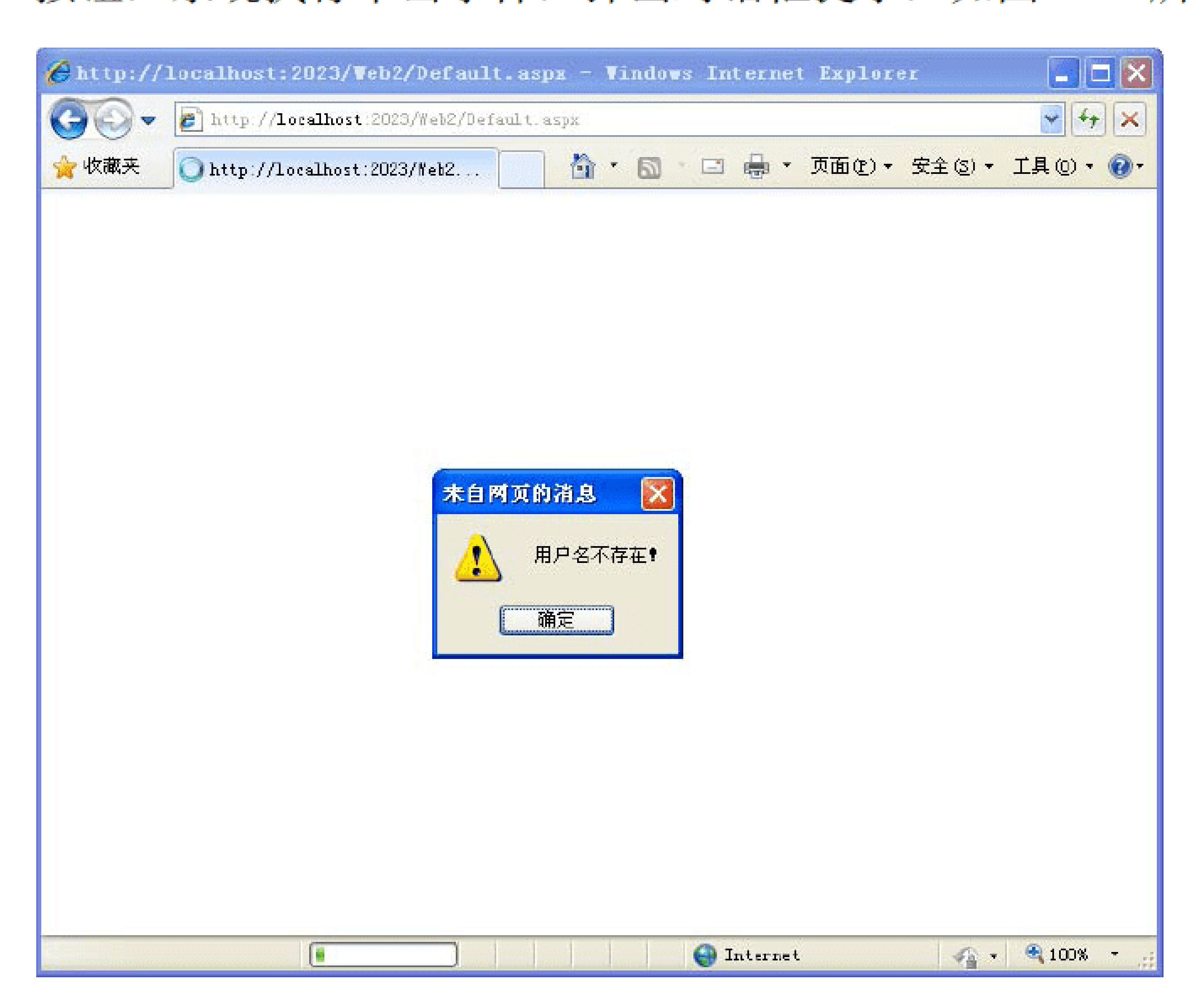


图 3-11　提示用户名不存在

5. 实例分析



源码解析：

实例在登录页面的登录事件触发的时候验证用户输入信息，并使用 Response 对象的 Write 方法向页面打印一个 HTML 的 Script 标签。

Script 标签内包含一个 alt 方法，该方法给用户提示执行结果信息。

后台程序中 Response 对象的 Write 方法将向页面首部打印参数内容。

3.4.2　输出XML内容

现在很多网站的开发都用到了 XML，它的功能不言而喻，能以结构化的方式描述各种类型的数据。XML 允许文档制作人员创建新的标记符，来更准确地描述数据。

现在很多网站中制作 Flash 广告时经常会用到 XML。我们可以动态地生成 XML 文件，以便于使 Flash 广告内容更容易管理和维护。

有时候我们的图片等资源并不想被人随意下载使用，所以经常隐藏资源的实际路径。在这里广告资源的路径在 XML 文件中存储着，我们只要将 XML 文件隐藏起来就行了。

1. 基础知识——ContentType属性和Charset属性

操作 XML 文件需要用到 XmlDocument 类。该类提供一个 Load(string xmlFilePath)方法，用来加载一个指定路径的 XML 文档。该类还提供了一个 InnerXml 属性，可以得到文档内的所有内容。

这里还要用到 Response 对象的两个属性：ContentType 属性用来设置整个页面文档的格式，Charset 属性用来设置文档的字符集。

这两个属性的语法如下：

```
Response.ContentType = "text/html";
Response.Charset = "utf-8";
```

Response 对象还提供了一个 End 方法，该方法可以将所有的缓冲发送到客户端，并结束当前输出流。

2. 实例描述

做一个博客，在首页用到了一个 Flash+XML 做的切换图片效果。现在想把 XML 文件隐藏起来，因为 Web 窗体容易控制和验证权限，所以这里使用一个 Web 窗体输出 XML 内容。

3. 实例应用

【例 3-8】输出 XML 内容。

(1)　创建一个 Web 项目。

(2)　添加网站内容，网站首页为 index.htm。

(3)　添加一个 Web 窗体，命名为 AdXML.aspx，以备输入 XML 文件之用。

(4)　在 AdXML.aspx 的后台代码的 Page_Load 方法里添加输出 XML 文件内容的代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    /* 初始化文件路径,
     * Server.MapPath 方法是获取指定站点虚拟路径对应的实际路径 */
    string filePath = Server.MapPath("~/") + "xml/toppicxml.xml";

    XmlDocument xd = new XmlDocument();     //初始化一个 XML 文档对象
    xd.Load(filePath);     //加载 XML 文件
```

```
    /* 设置文档格式和字符集 */
    Response.ContentType = "text/xml";
    Response.Charset = "utf-8";

    Response.Write(xd.InnerXml);    //向页面输出 XML 文档内容
    Response.End();    //结束输出流
}
```

(5) 在网站首页添加广告展示的 Flash 对象。在引用 XML 路径的地方将 XML 路径改为 AdXML.aspx。

代码如下：

```
<param name="movie" value="flash/toppic.swf?data=AdXML.aspx" />
<embed src="flash/toppic.swf?data=AdXML.aspx" ... />
```

(6) 保存所有文件。

4. 运行结果

运行项目，访问 index.htm 页面。

访问结果如图 3-12 所示。



图 3-12　Flash+XML

访问 AdXML.aspx 页面，结果如图 3-13 所示。

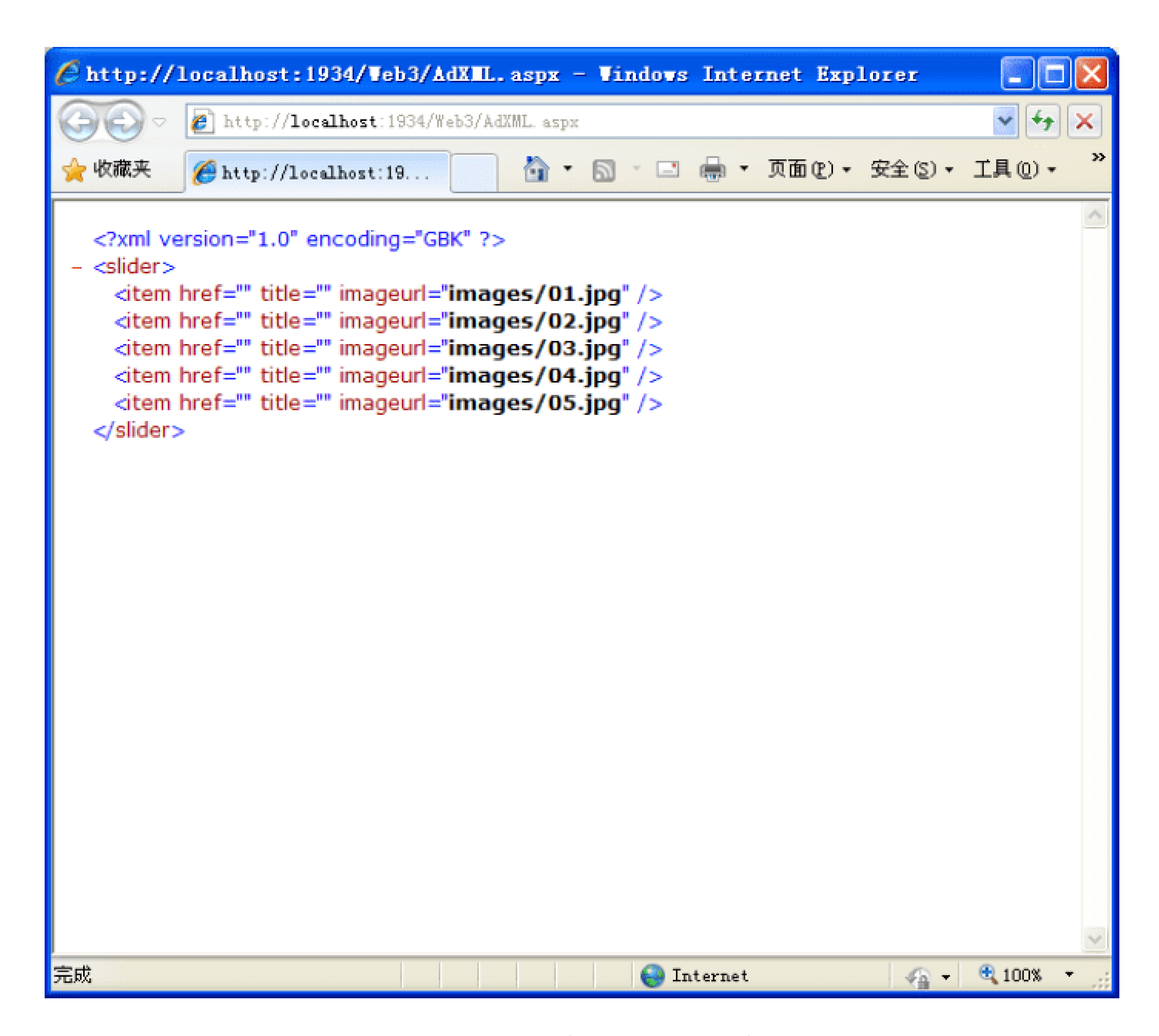


图 3-13　输入XML内容

5. 实例分析



源码解析：

实例创建了一个 Web 窗体，用于输出 XML 文档。在该 Web 窗体的 Page_Load 事件处理程序中创建了一个用以操作 XML 文件的 XmlDocument 类的实例，并加载指定的 XML 文件。同时设置该文档格式为 text/xml，使用的字符集为 urf-8。然后将 XmlDocument 对象中的 XML 文件内容使用 Response 对象的 Write 方法输入到响应流中，并发送响应结束响应流。

3.4.3　输出图像

Response 对象除了能够向客户端浏览器输出文本信息以外，还可以输出其他任意类型的数据到浏览器中，例如图像。

ASP.NET 支持将图像文件以二进制流的形式返回给客户端。这样不仅可以隐藏图像文件的实际地址，而且可以对图像进行处理后再返回。例如最常见到的验证码就是使用代码生成一个图片文件，然后再把图片的二进制流输出到客户端。还有图像的水印效果，也可以把图片加载到内存，处理完以后再发送给客户端。

这里我们就简单地演示一下如何向客户端输出一张已存在的图像。

1. 基础知识——BinaryWrite方法和FileStream类

读取文件需要使用 System.IO 命名空间下的 FileStream 类。

FileStream 类有十多个构造方法，这里我们使用其中的一个来构建该类的实例，如下：

```
FileStream(string path, FileMode mode);
```

该构造方法有两个参数。

- path：是指初始化文件流对象指定的文件路径。
- mode：是指访问该文件的方式。

mode 参数是一个 FileMode 枚举，有以下 6 个选项：CreateNew、Create、Open、OpenOrCreate、Truncate、Append。

FileStream 类还提供了一个 Read 方法，可以从流中读取指定的字节块并写入指定的二进制数组中。格式如下：

```
Read(byte []array, int offset, int count);
```

该方法有 3 个参数：

- array：该参数是一个二进制数组，用于接收二进制数据。
- offset：从流中读取字节的起始位置。
- count：从流中读取数据的长度。

FileStream 类还提供了一个 Close 方法来关闭 FileStream 流对象，释放文件资源。

另外，向响应流中输入二进制数据需要使用 BinaryWrite 方法，该方法只需要一个参数，即一个二进制数组即可。该方法的功能是将二进制数组中的数据输入响应流中。

2. 实例描述

刚才说了 ASP.NET 中的 Web 窗体能将图像以二进制的形式返回给客户端，这里作者的网站根目录下有一个名为 sg.jpg 的图片文件，现在需要将其用 Web 窗体显示出来，并作为页面 img 标签的图像源展示给用户。

3. 实例应用

【例 3-9】输出图像。

(1) 新建一个 Web 项目，根目录下保存一个图片文件 sg.jpg。

(2) 新建一个 Web 窗体，命名为 Picture.aspx。

(3) 打开该窗体的后台程序文件，在 Page_Load 方法里添加代码。如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    string picPath = Server.MapPath("sg.jpg");    //初始化图像物理地址
    /* 初始化文件流，用以读取图像 */
    FileStream fs = new FileStream(picPath, FileMode.Open);
    if (fs != null)
    {
        /* 读取图像的二进制数据，保存到二进制数组 data 中 */
        byte []data = new byte[(int)fs.Length];
        fs.Read(data, 0, (int)fs.Length);
        fs.Close();    //关闭文件流
        Response.ContentType = "image/jpeg";    //设置页面输出类型
        Response.BinaryWrite(data);    //输出二进制数据
```

```
    }
}
```

(4)　打开 Default.aspx 页面，在页面中添加一个 img 标签，如下所示：

```
<img alt="" src="Picture.aspx" class="pic" />
```

(5)　保存所有文件。

4．运行结果

运行项目，访问 Picture.aspx 页面，结果如图 3-14 所示。

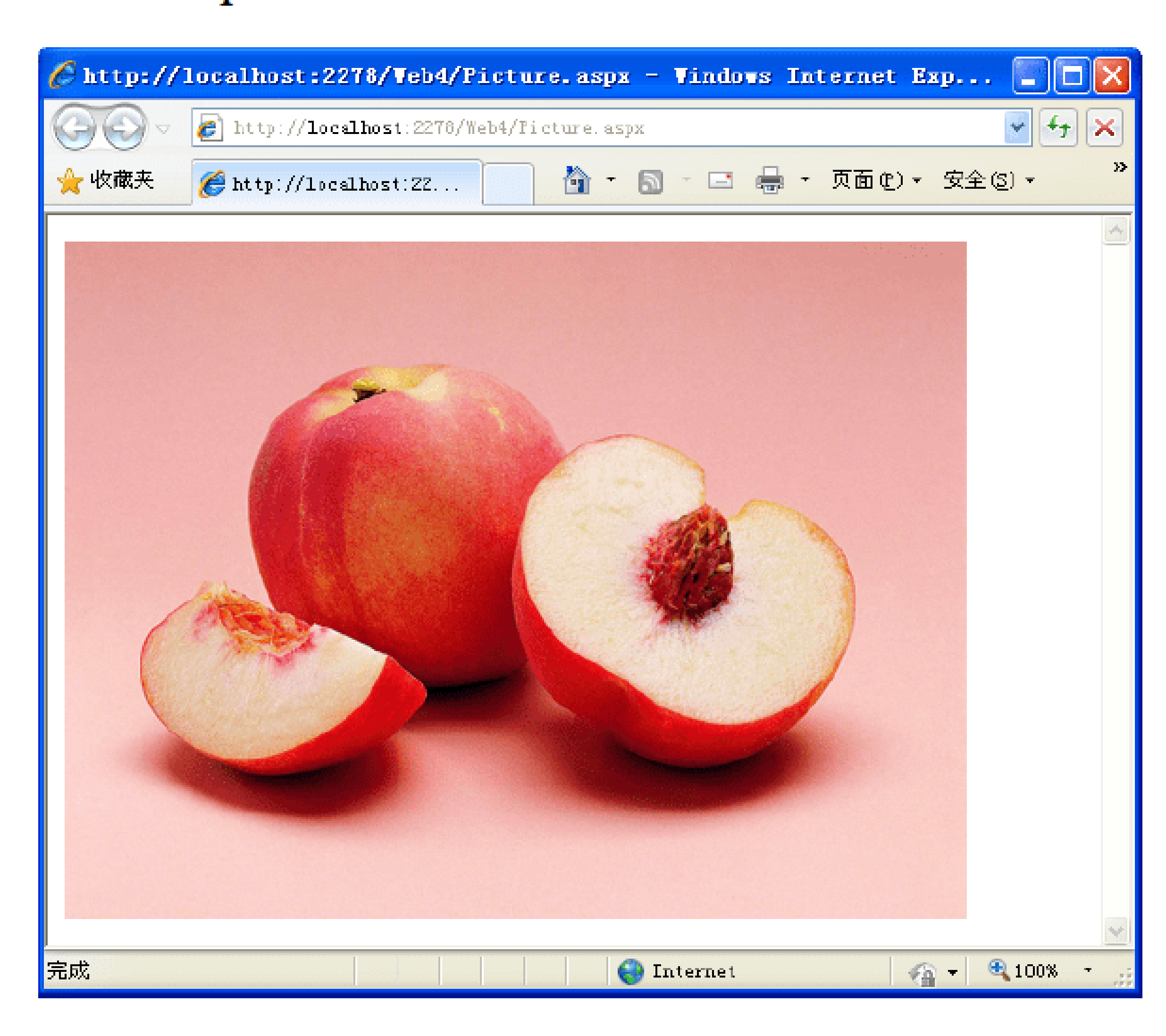


图 3-14　输入图像

Default.aspx 页面使用 img 标签引用了该页面，我们访问一下 Default.aspx 页面，结果如图 3-15 所示。



图 3-15　以img标签引用该图像

5. 实例分析



源码解析：

实例使用 FileStream 读取该图像文件，并将图像的二进制信息全部读取到一个二进制数组，并将该数组使用 Response 对象的 BinaryWrite 方法发送到响应流中。

我们看到这样可以使用 Web 页面向客户端输出一个图像文件，可以像使用一般图像文件一样使用该 Web 页面。

3.4.4 页面执行跳转

在 JavaScript 中，通过设置 Location 对象可以控制页面重定向到另一个 URL，非常方便。

但是在执行 ASP.NET 程序的时候通常也会因为某些原因，需要使页面重定向到另一个地址。当然这个操作可以在页面中打印出一行 JavaScript，内容就是设置页面 Location 对象的值为另一个 URL 地址。这种方法可以实现，但是毕竟太费事，不好用。

Response 对象提供了一个 Redirect 方法来重定向 URL，我们可以在 ASP.NET 后台程序中需要执行跳转的地方直接调用该方法即可。

1. 基础知识——Redirect方法

System.Web.HttpResponse 类为 Redirect 方法提供了两个重载。

- Redirect(string url)：跳转到指定 URL。该方法只用一个 URL 作为路径，使该请求自动跳转到指定的 URL。
- Redirect(string url, bool endResponse)：跳转到指定的 URL。该方法使用两个参数，参数 url 为目标 URL，参数 endResponse 指示当前响应是否结束。

2. 实例描述

很多系统登录功能都会专门设计一个登录页面，当用户登录不成功的时候提示用户重新登录，当用户登录成功的时候执行跳转操作，将页面重定向到登录成功后的管理首页。

刚好前些天做了一个网站，网站后台登录就用到这个功能。我们一起来看一下。

3. 实例应用

【例 3-10】 页面执行跳转。

(1) 新建一个 Web 项目。

(2) 以 Default.aspx 为登录页面，添加一个用户名文本框 txtUsername，添加一个密码框 txtPassword，添加一个提交请求的命令按钮 imgbtnSubmit，添加一个用于提示操作结果的标签控件 lblResult。

(3) 给命令按钮添加单击事件。修改事件处理程序如下：

```
protected void imgbtnSubmit_Click(object sender, ImageClickEventArgs e)
{
```

```
    /* 验证登录 */
    if (this.txtPassword.Text == "admin" && this.txtUsername.Text == "admin")
    {
        Response.Redirect("index.htm");    //跳转到管理页面
    }
    else
    {
        this.lblResult.Text = "用户名或密码错误!";
    }
}
```

这里 index.htm 页面即是管理后台主界面。

(4) 保存，完成实例。

4. 运行结果

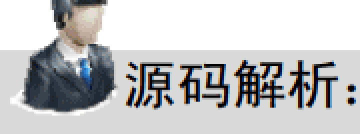
运行项目，访问 Default.aspx 页面。结果如图 3-16 所示。



图 3-16　登录界面

输入用户名 admin 和密码 admin，单击“登录”按钮，浏览器跳转到管理首页，如图 3-17 所示。

5. 实例分析



源码解析：

该实例创建了两个文本框控件接收用户输入，根据用户输入判断是否登录成功，如果成功则使用 Response 对象的 Redirect 方法跳转到目标页面，不成功则在页面标签中输出提示信息。



图 3-17　管理首页

3.5　获取服务器端信息

在处理用户请求的时候，往往还需要获取一些服务器的信息，比如获取应用程序的物理路径或服务器的计算机名称，或者需要进行一些编码解码之类的辅助操作。在 ASP.NET 中，其特有的 Server 对象封装了这些辅助功能，可以实现对用户请求的一些辅助性的操作。

Server 对象是 System.Web.HttpServerUtility 类的对象。系统将其作为一个只读的公有属性内置在 Page 类中，我们可以在任何一个 Page 类中直接使用。

Server 对象的常用方法和属性如表 3-9 所示。

表 3-9　Server对象的常用方法和属性

名　称	说　明
MachineName 属性	获取服务器的计算机名称
ScriptTimeout 属性	获取和设置请求超时值(以秒计)
ClearError 方法	清除前一个异常
CreateObject 方法	创建 COM 对象的服务器实例， 该 COM 对象由对象的程序标识符(ProgID)标识
CreateObjectFromClsid 方法	创建 COM 对象的服务器实例，该对象由对象的类标识符(CLSID)标识
Execute 方法	在当前请求的上下文中执行指定虚拟路径的处理程序
GetLastError 方法	返回前一个异常
HtmlDecode 方法	对 HTML 编码的字符串进行解码，并返回已解码的字符串
HtmlEncode 方法	对字符串进行 HTML 编码并返回已编码的字符串

续表

名　称	说　明
MapPath 方法	返回与 Web 服务器上的指定虚拟路径相对应的物理文件路径
Transfer 方法	对于当前请求，终止当前页的执行，并使用指定页的 URL 路径来开始执行一个新页
TransferRequest 方法	异步执行指定的 URL
UrlDecode 方法	对字符串进行 URL 解码并返回已解码的字符串
UrlEncode 方法	对字符串进行 URL 编码，并返回已编码的字符串
UrlPathEncode 方法	对 URL 字符串的路径部分进行 URL 编码并返回编码后的字符串
UrlTokenDecode 方法	将 URL 字符串标记解码为使用 64 位二进制数字的等效字节数组
UrlTokenEncode 方法	将一个字节数组编码为使用 Base 64 编码方案的等效字符串表示形式，Base 64 是一种适于通过 URL 传输数据的编码方案

3.5.1　获取网站执行目录

我们经常要操作 Web 应用程序所在目录内的一些文件，操作的时候往往要用到该文件的绝对路径。

Server 对象提供了一个 MapPath 方法，可以很方便地使用该方法获得指定相对路径的物理地址。

其实在前面的几个实例中我们已经简单地使用过它了，本小节再来详细了解一下。


视频教学： 光盘/videos/3/MapPath_Method.avi　　长度：5 分钟

1. 基础知识——MapPath方法

Server 对象的 MapPath 方法声明如下：

```
public string MapPath(string path);
```

该方法没有其他重载。唯一的参数是一个虚拟路径，可以有以下几种形式。

- “~”或“.”：该方式是 ASP.NET 指定站点根目录的独有方式，它返回站点根目录的物理地址。
- “virtualPath”：虚拟路径，可以是一个目录，也可以是文件或者目录加文件。例如 image/banner.jpg。



这里的参数 path 所指的路径不可以是路径间隔符——斜线开头。正反斜线都不行。

该方法返回一个字符串对象，路径为当前站点的物理路径加上参数所指的虚拟路径以后的物理路径。结构为“盘符:\站点路径\虚拟路径”。

2. 实例描述

上面我们已经使用过该方法，本实例就直接列出该方法参数的几种形式，我们一起来查看

一下执行结果。

3. 实例应用

【例 3-11】获取站点执行目录。

(1) 新建一个 Web 项目。

(2) 修改 Default.aspx 页面的后台代码，修改 Page_Load 方法如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write("<br>" + Server.MapPath("~/"));
    Response.Write("<br>" + Server.MapPath("."));
    Response.Write("<br>" + Server.MapPath("images"));
    Response.Write("<br>" + Server.MapPath("logo.gif"));
    Response.Write("<br>" + Server.MapPath("image/banner.jpg"));
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面。访问结果如图 3-18 所示。

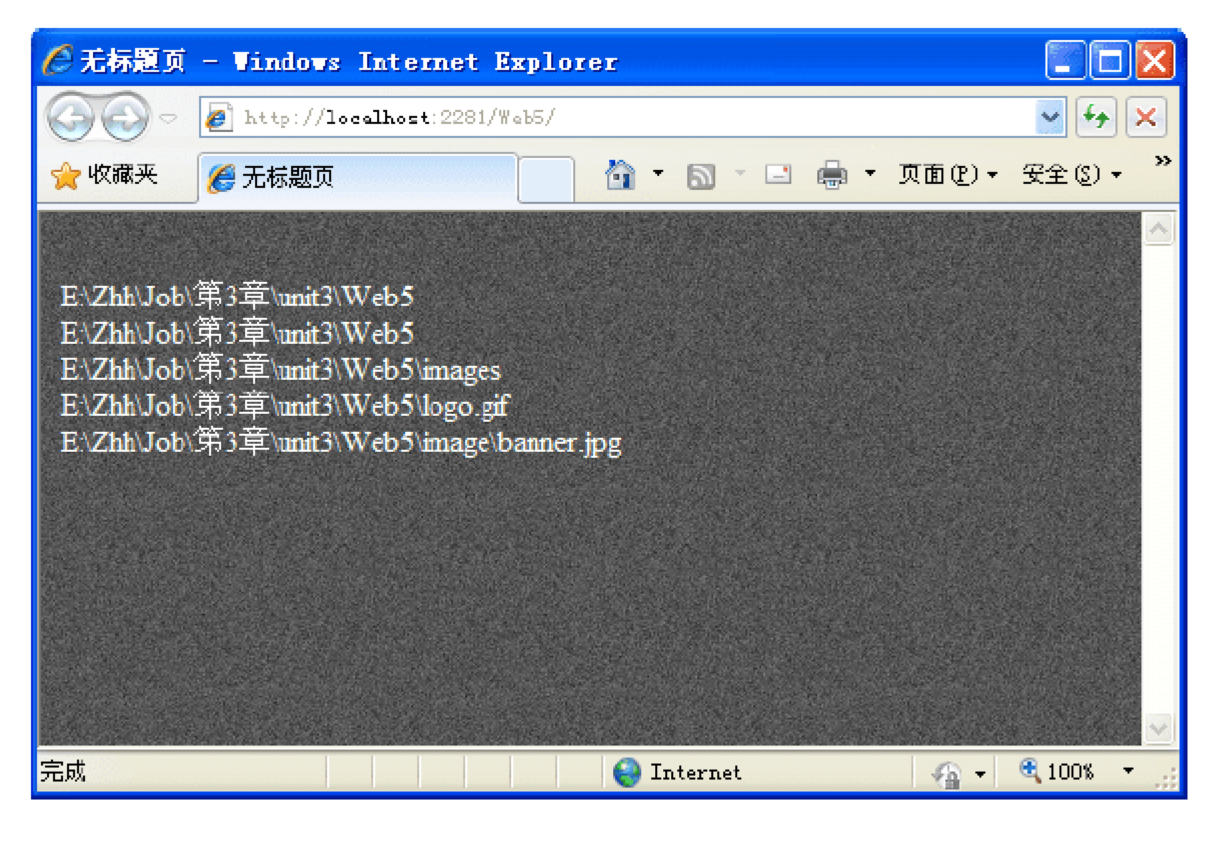


图 3-18 返回网站执行目录

5. 实例分析



源码解析：

实例分别使用 Response 对象的 Write 方法向页面打印了 Server 对象的 MapPath 方法获取的站点虚拟路径所对应的物理路径。

从实例结果可以看到，Server 对象的 MapPath 方法可以定位站点下的所有目录和文件。所以这为我们操作网站上的文件提供了很大的方便。

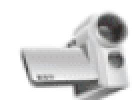
3.5.2 执行页面转发

前面讲过使用 Response.Redirect 方法可以使浏览器跳转到新的 URL。这种方式就好像是有

人来找你办事，你办不了或者不想办，你就告诉来办事的那个人：你应该去找某某某办这件事，这件事他负责。

不过有些事的确是在我们职责范围内的，不过因为这件事需要几个人协调，或者你需要后方技术支持。别人再来找你办事，你就不能把他甩到别人那里去了，你需要亲自找人帮忙，办完这件事以后把结果返回给来办事的人。这样，这个来找你办事的人就会认为是你帮他办好的事，而不知道你还找了其他的人。

在 ASP.NET 中也提供了这种情况的实现机制，那就是 Server 对象的 Transfer 方法。



视频教学：光盘/videos/3/Transfer_Method.avi　　长度：5 分钟

1. 基础知识——Transfer方法

Server 对象的 Transfer 方法用以将用户请求转发到本网站上的指定虚拟路径上面，地址栏信息和 QueryString 集合以及 Form 集合都可以保存并让新页面接着使用。

该方法有 3 个重载，具体说明如下。

- Transfer(string path)：对于当前请求，终止当前页的执行，并使用指定页的 URL 路径来开始执行一个新页。参数 path 是服务器上要执行的新页的虚拟路径。
- Transfer(IHttpHandler handler, bool preserveForm)：终止当前页的执行，然后使用一个实现 System.Web.IHttpHandler 接口的自定义 HTTP 处理程序开始新请求的执行，并指定是否要清除 System.Web.HttpRequest.QueryString 和 System.Web.HttpRequest.Form 集合。参数 handler 实现 System.Web.IHttpHandler 以便向其传输当前请求的 HTTP 处理程序。bool 型参数 preserveForm 指定是否要清除以上所说的两个集合。
- Transfer(string path, bool preserveForm)：终止当前页的执行，并使用指定页的 URL 路径来开始执行一个新页。指定是否清除 System.Web.HttpRequest.QueryString 和 System.Web.HttpRequest.Form 集合。参数 path 是服务器上要执行的新页的虚拟路径。bool 型参数 preserveForm 指定是否要清除以上所说的两个集合。

2. 实例描述

很多国际化的网站通常是多语言版的，我们访问同一个 URL，在不同地区的人就会看到不同语言的内容。

这种功能的比较简单的一种实现方案就是使用 URL 转发，这样地址栏内容保持不变，就仿佛我们不同的地域访问的是同一个页面。

本实例就在访问网站的时候测试访客主机的语言版本，给访客返回相应的页面。为了便于测试，也可以接收请求参数设置指定语言。

3. 实例应用

【例 3-12】执行页面转发。

(1) 新建一个 Web 项目。

(2) 拷入项目内容文件。这里至少需要一个 cn_index.html 和一个 en_index.html。

(3) 在 Default.aspx 页面的后台 Page_Load 方法里添加代码如下：

```
protected void Page_Load(object sender, EventArgs e)
```

```
{
    string language = Request.UserLanguages[0];    //获取当前客户端语言

    //如果请求数据中有语言设置，设置为所请求的语言
    if (Request["language"]!=null && Request["language"]!="")
    {
        language = Request["language"];
    }

    //如果客户端语言为中文，返回中文页面，反之返回英文页面
    if (language == "zh-cn")
    {
        Server.Transfer("cn_index.html");
    }
    else
    {
        Server.Transfer("en_index.html");
    }
}
```

(4) 保存。

4. 运行结果

运行项目。访问 Default.aspx 页面，返回结果如图 3-19 所示。



图 3-19 中文页面

该路径返回中文页面，我们在地址栏 URL 后面以 GET 方式加入一个 language 参数，值为 en。

执行请求，返回结果如图 3-20 所示。

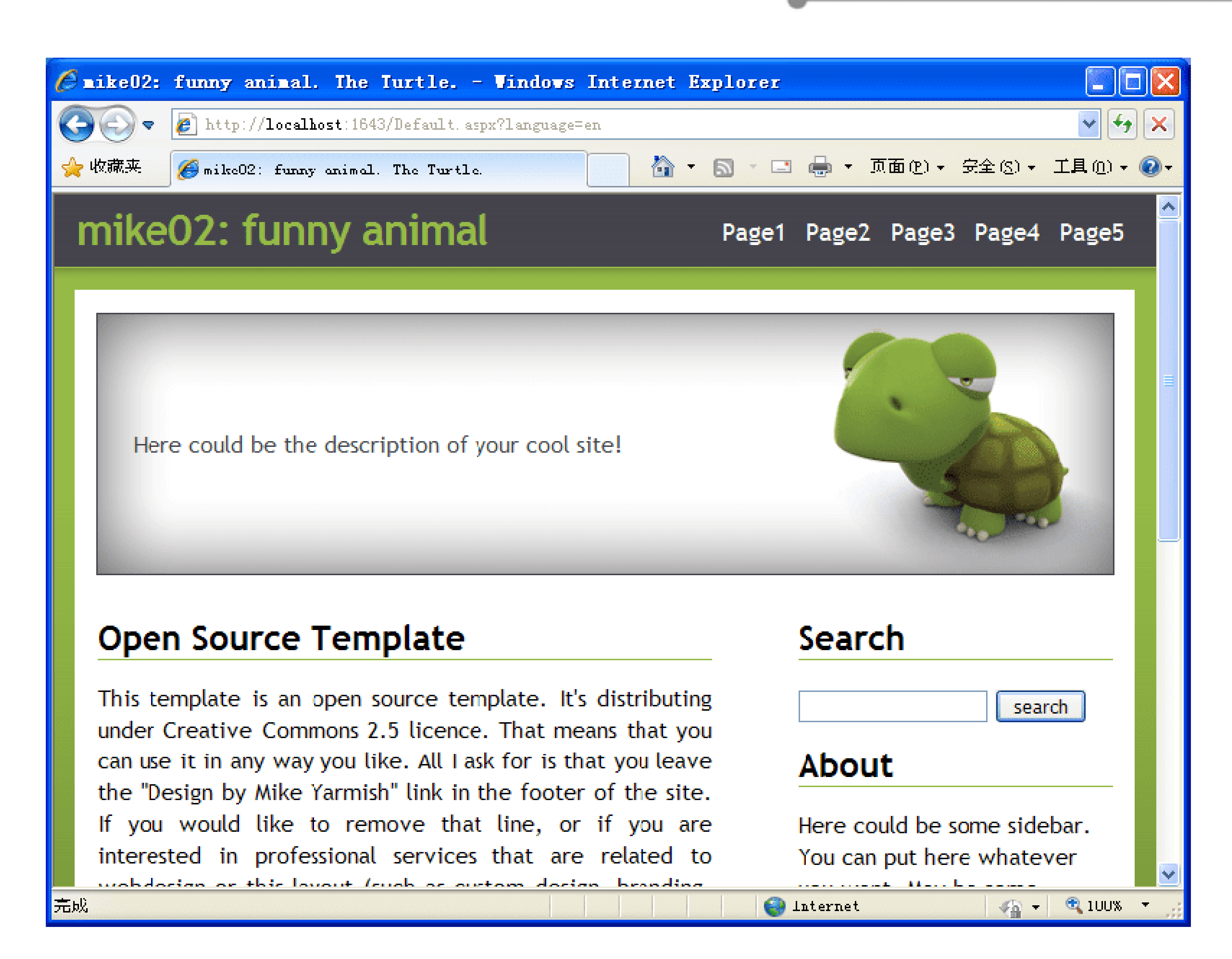


图 3-20　英文页面

5. 实例分析



源码解析：

本实例首先使用 Request 对象的 UserLanguages 属性获取客户端语言数组，再判断用户请求中有没有设置语言的参数，如果有，优先使用请求参数中的语言，然后判断语言是否为中文，如果是中文就将请求转发到中文页面，否则将请求转发到英文页面。

3.5.3　对HTML进行编码和解码

由于 HTML 是一种由符号标记的语言，所以该语言占用了一些表示的符号。而页面随时需要表示这些符号，所以 HTML 将一些被占用的符号或一些特殊功能的符号使用了一些特殊的方法标记，以便展示。这些方法就是 HTML 编码。

在进行 HTML 文本处理的时候，通常要对 HTML 文本进行编码和解码的操作。在 ASP.NET 中，Server 对象提供了 HtmlEncode 和 HtmlDecode 两个方法实现对 HTML 文本进行编码和解码的操作。


视频教学：光盘/videos/3/Encode_Decode.avi　　　长度：8 分钟

1. 基础知识——HtmlEncode方法和HtmlDecode方法

(1) Server 对象的 HtmlEncode 方法对 HTML 文本进行编码。该方法有两个重载。

- string HtmlEncode(string s)：对字符串进行 HTML 编码并返回已编码的字符串。参数 s 是要编码的字符串。

- void HtmlEncode(string s, TextWriter output)：对字符串进行 HTML 编码，并将结果输出发送到文本输出流。参数 s 是要编码的字符串，参数 output 是 System.IO.TextWriter 类型的文本输出流。

(2) Server 对象的 HtmlDecode 方法将对 HTML 编码的字符串进行解码操作。该方法也有两个重载。

- string HtmlDecode(string s)：对 HTML 编码的字符串进行解码，并返回已解码的字符串。参数 s 是要解码的字符串。
- void HtmlDecode(string s, TextWriter output)：对 HTML 编码的字符串进行解码，并将结果输出发送到文本输出流。参数 s 是要解码的字符串，参数 output 是 System.IO.TextWriter 类型的文本输出流。

2. 实例描述

对 HTML 文本进行编码和解码通常用于对页面内容的展示效果的控制。

本实例我们就简单地输入一些 HTML 文本，并对其进行编码和解码操作。

3. 实例应用

【例 3-13】对 HTML 进行编码和解码。

(1) 新建一个 Web 项目，并添加相应的资源文件(如图片、样式表等)。

(2) 编辑 Default.aspx 页面，在页面内容区添加如下代码：

```
<asp:TextBox ID="txtSourceCode" runat="server" TextMode="MultiLine"
  Width="300" Height="60">
</asp:TextBox>
<br />
<asp:Button ID="btnEnCode" runat="server" Text="EnCode"
  onclick="btnEnCode_Click" />
<br />
未进行编码直接在页面展示：
<br />
<asp:Label ID="lblSource" runat="server"></asp:Label>
<br />
编码过后在页面展示：
<br />
<asp:Label ID="lblEnCode" runat="server"></asp:Label>
<br />
编码过后的文本：
<br />
<asp:TextBox ID="txtTargetCode" runat="server" TextMode="MultiLine"
  Width="300" Height="60">
</asp:TextBox>
<br />
<asp:Button ID="btnDeCode" runat="server" Text="DeCode" onclick="btnDeCode_Click" />
<br />
解码后在页面展示：
<br />
<asp:Label ID="lblDeCode" runat="server"></asp:Label>
```

第一个文本框 txtSourceCode 接收输入的一段 HTML 文本，标签 lblSource 直接将输入的那段 HTML 文本展示到页面上，标签 lblEnCode 将编码过的代码文本展示到页面上，文本框 txtTargetCode 显示编码过的字符信息，标签 lblDeCode 将文本框 txtTarget 里的文本重新解码成的 HTML 文本展示出来。

(3) 我们需要在单击 EnCode 的时候对文本框 txtSourceCode 里的内容进行编码。这里在设计视图下双击添加按钮 btnEnCode 的单击事件，并修改事件处理程序，如下所示：

```
protected void btnEnCode_Click(object sender, EventArgs e)
{
    string source = this.txtSourceCode.Text;    //获取文本框中的值
    this.lblSource.Text = source;    //直接展示输入的内容
    string enCode = Server.HtmlEncode(source);    //编码
    this.lblEnCode.Text = enCode;    //在页面展示编码过后的内容
    this.txtTargetCode.Text = enCode;  //输出编码过后的文本供查看
}
```

(4) 另外我们还要添加 DeCode 按钮的单击事件，该事件执行解码操作。在后台代码中修改事件处理程序如下：

```
protected void btnDeCode_Click(object sender, EventArgs e)
{
    string source = this.txtTargetCode.Text;    //获取文本框中的值
    string deCode = Server.HtmlDecode(source);    //执行解码
    this.lblDeCode.Text = deCode;    //展示解码后的内容
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面。

访问结果如图 3-21 所示。

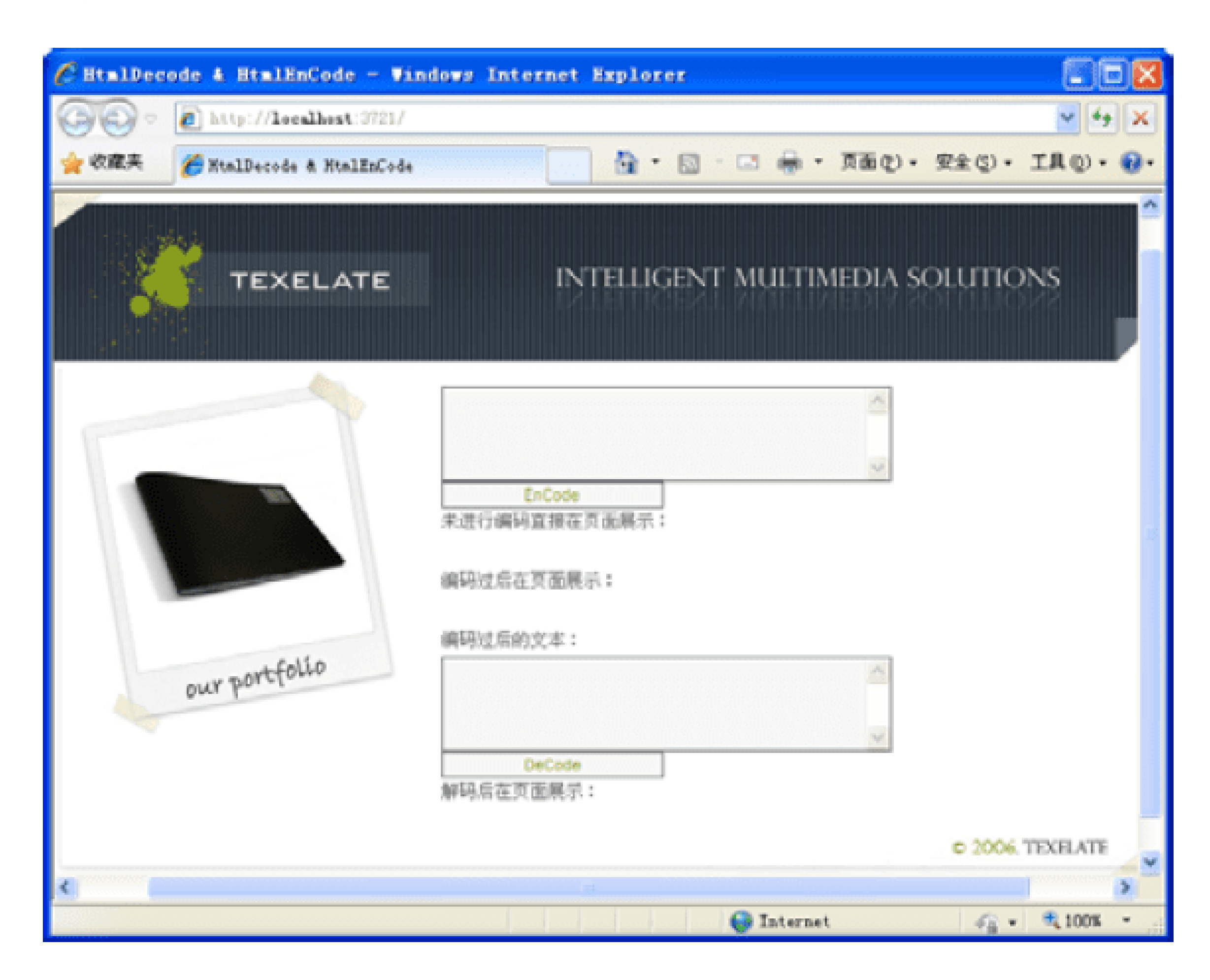


图 3-21　程序界面

我们在第一个文本框中输入如下一行数据：

```
<h1>Html DeCode && Html EnCode</h1>
```

然后单击 EnCode 按钮，执行结果如图 3-22 所示。

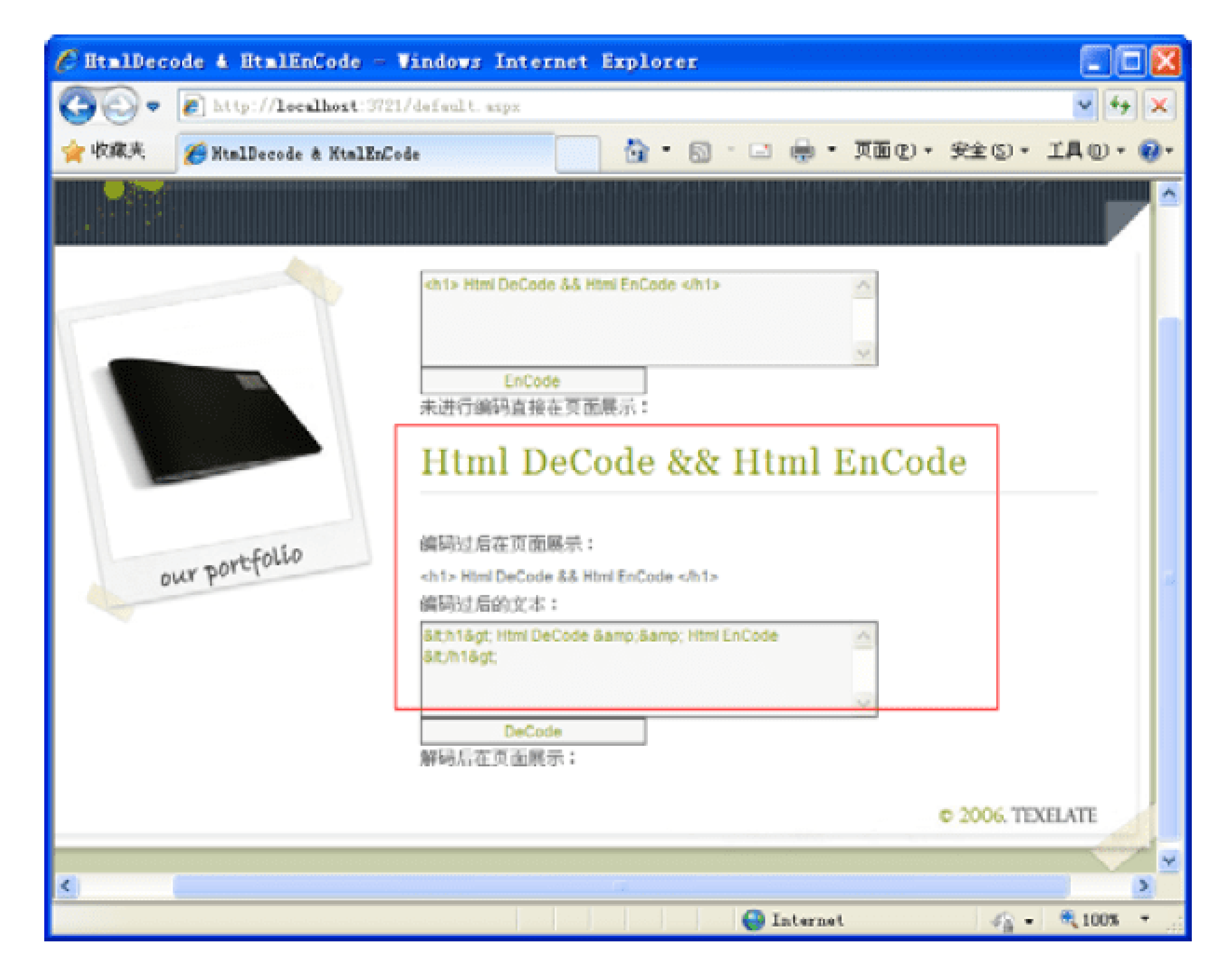


图 3-22　执行编码操作

编码过后的文本已经添加到了第二个文本框中。直接单击 DeCode 按钮，执行结果如图 3-23 所示。

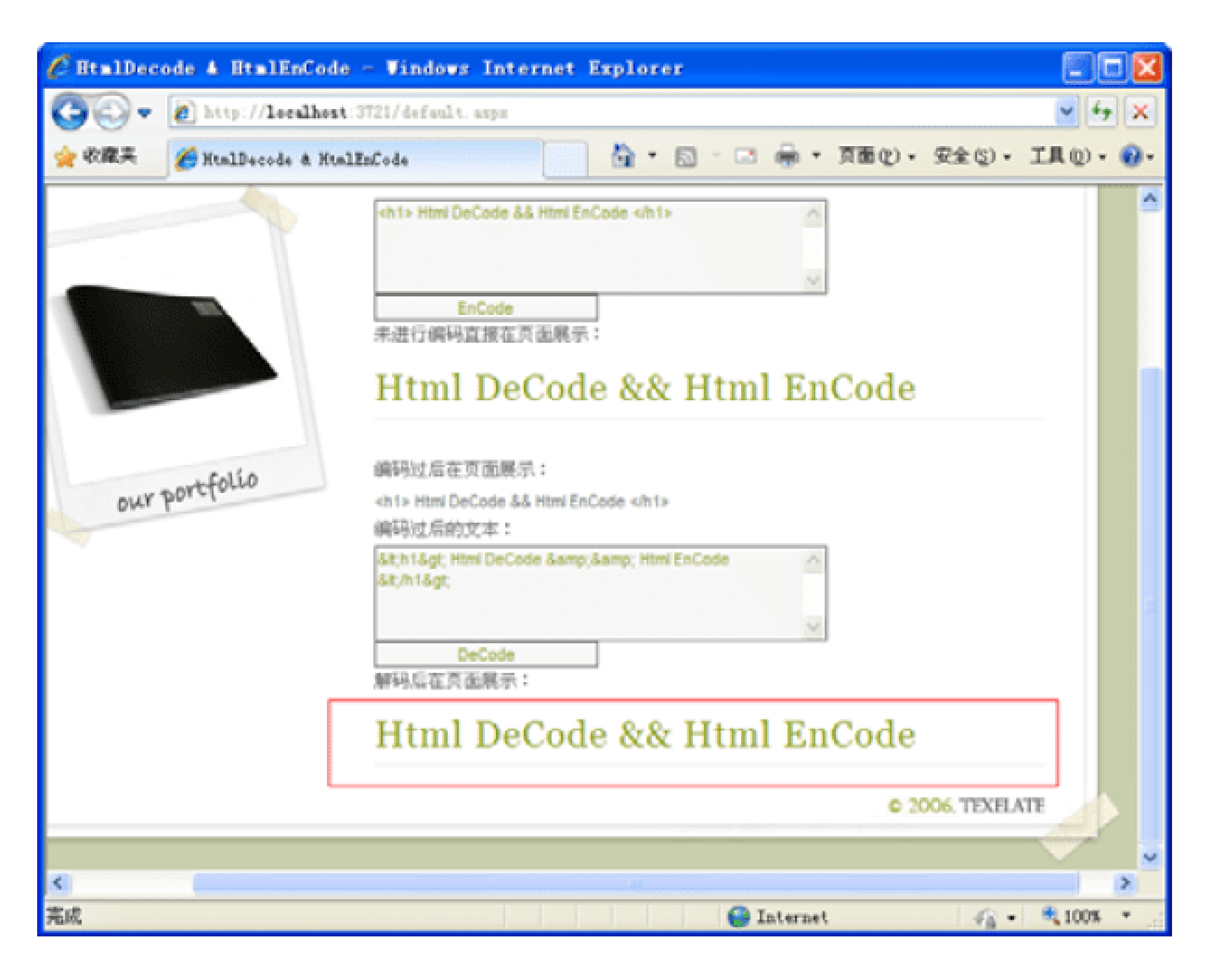


图 3-23　执行解码操作

5. 实例分析



源码解析：

实例首先使用 Server 对象的 EnCode 方法对第一个文本框中输入的内容进行编码，并把编码过后的文本展示给用户，并存入第二个文本框，接下来又使用 Server 对象的 DeCode 方法对第二个文本框里的已经编码过的文本进行解码，展示到页面上。

3.5.4 URL汉字编码和解码

以 Get 方式传参来访问 URL 是非常常用的一种方式。

不过我们经常会需要将一些中文文字作为参数值提交给 Web 服务器，Web 服务器也常常把 URL 上的文本按默认的编码方式解码，所以我们经常会遇到类似于“ ǹ▯ ”这样的一些乱码数据。

为了解决这个问题，业界统一对一些中文文本等特殊的符号规定了一套编码，在地址栏请求的时候按指定规则进行编码，在服务器端解析的时候按这个规则进行解码，这样就能准确地得到提交的中文文本内容。

 **视频教学：** 光盘/videos/3/Url_code.avi　　长度：6 分钟

1. 基础知识——UrlEncode方法和UrlDecode方法

Server 对象中 UrlEncode 方法负责对文本内容进行编码，UrlDecode 方法负责对编码过的文本进行解码。

(1) UrlEncode 方法有两个重载。

- string UrlEncode(string s)：对字符串进行 URL 编码，并返回已编码的字符串。参数 s 是要进行 URL 编码的文本。
- void UrlEncode(string s, TextWriter output)：对字符串进行 URL 编码，并将结果输出发送到文本输出流。参数 s 是要进行 URL 编码的文本，参数 output 是接收编码结果的文本流，该参数是 System.IO.TextWriter 类对象。

(2) UrlDecode 方法也有两个重载。

- string UrlDecode(string s)：对字符串进行 URL 解码并返回已解码的字符串。参数 s 是要进行 URL 解码的文本。
- void UrlDecode(string s, TextWriter output)：对在 URL 中接收的 HTML 字符串进行解码，并将结果输出发送到文本输出流。参数 s 是要进行 URL 解码的文本，参数 output 是接收解码结果的文本流，该参数是 System.IO.TextWriter 类的对象。

2. 实例描述

Get 方式传参使用最多的是在执行查询的时候，往往让查询条件使用 Get 方式传递给服务器，这样也更容易让用户直接保存地址栏内的信息，更容易标记与传播。

本实例我们简单地模拟一下网站搜索功能的实现。

在用户输入搜索条件，单击“搜索”按钮的时候，将搜索条件进行 URL 编码并合并 URL 跳转到指定路径。在接收到 URL 编码的时候，先进行解码，再将解码的结果展示在页面上。

3. 实例应用

【例 3-14】 URL 汉字编码和解码。

(1) 新建一个项目。

(2) 导入站点资源文件(图片、样式表等)。

(3) 添加用于搜索功能的文本框 txtKey 和命令按钮 btnSearch。代码如下：

```
<asp:TextBox ID="txtKey" runat="server"></asp:TextBox>
<asp:Button ID="btnSearch" runat="server" Text="搜索" onclick="btnSearch_Click" />
```

(4) 为“搜索”按钮添加单击事件，修改事件处理程序，代码如下：

```
protected void btnSearch_Click(object sender, EventArgs e)
{
    string str = this.txtKey.Text;    //获取关键字
    str = Server.UrlEncode(str);    //编码
    Response.Redirect("Default.aspx?key=" + str);    //跳转
}
```

(5) 在页面添加一个标签，用于展示从 URL 中接收到的值。代码如下：

```
<div>请求的 Key 值为：<asp:Label ID="lblKey" runat="server"></asp:Label></div>
```

(6) 在页面加载的时候获取 URL 的值，并进行解码。修改 Page_Load 方法为：

```
protected void Page_Load(object sender, EventArgs e)
{
    string str = Request.QueryString["key"];    //获取 URL 值
    str = Server.UrlDecode(str);    //解码
    this.lblKey.Text = str;    //展示到页面上
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面。

访问结果如图 3-24 所示。



图 3-24　汉字编码

在“搜索”文本框中输入“这是关键字”然后单击“搜索”按钮，结果如图 3-25 所示。

图 3-25　汉字解码

这里我们注意一下地址栏里编码的内容和页面上解码的内容(图中红框所示)。

5. 实例分析



源码解析：

实例使用 Server 对象的 UrlEncode 方法对文本框的内容进行编码，合并参数后跳转到 Default.aspx 页面，在 Default.aspx 页面的 Page_Load 方法里对 URL 请求的值使用 Server 对象的 UrlDecode 方法进行解码并展示到页面上。

3.6　统计网站在线人数

在一些互动型的 Web 应用程序中(比如 BBS、社区等)，管理员通常会想让用户了解当前在线用户的一些信息，以达到更好的互动效果。

浏览过不少的 BBS，了解到现在使用最多的功能就是在线人数了。

我们知道，在 Web 应用程序中，用户的访问是无状态的，所以我们没有办法对用户的状态做持久、及时的监控。

不过为了保持用户的状态，当用户访问 Web 应用程序时，Web 应用程序将为每个用户创建一个保持会话信息的 Session 对象，我们可以使用 ASP.NET 全局应用程序来监控 Session 对象的创建和销毁事件，来确认用户的登录和注销。



视频教学：光盘/videos/3/ApplicationObject.avi　　**长度：**8 分钟

3.6.1　基础知识——Global.asax和Application对象

ASP.NET 提供了一个全局应用程序类，用来监控应用程序、会话、用户请求等对象的运

行状态。

全局应用程序类存储在程序根目录下的 Global.asax 文件中，类名为 Global，是一个 System.Web.HttpApplication 类的子类。

该类自动实现了一系列方法，如表 3-10 所示。

表 3-10　Global默认方法

方 法 名	说　明
Application_AuthenticateRequest	认证请求的时候触发该方法
Application_BeginRequest	开始一个新的请求时触发该事件，每次 Web 服务器被访问都执行该方法
Application_End	应用程序结束事件。在这里可以做一些停止应用程序时的善后工作
Application_Error	应用程序出现错误时触发，可以做一些错误处理操作
Application_Start	应用程序启动事件，可以在这里做一些全局对象初始化操作
Session_End	结束一个会话时执行该方法。这里是指在 Session 对象销毁时触发
Session_Start	创建一个会话时执行该方法。这里是指在 Session 对象创建时触发

实现在线人数统计时，我们可以使用 Session_End 方法和 Session_Start 方法进行监控。不过还需要在应用程序中把统计结果进行全局保存，这里要用到 Application 对象。

Application 可以在多个请求、连接之间共享公用信息，也可以在各个请求连接之间充当信息传递的管道。使用 Application 对象来保存希望传递的变量。由于在整个应用程序生存周期中，Application 对象都有效，所以在不同的页面中都可以对其进行存取，就像使用全局变量一样方便。

Application 对象是 System.Web.HttpApplicationState 类的实例。该类提供的属性和方法如表 3-11 所示。

表 3-11　Application属性和方法

名　称	说　明
AllKeys 属性	获取 HttpApplicationState 集合中的访问键
Count 属性	获取 HttpApplicationState 集合中的对象数
Add 方法	将新的对象添加到 HttpApplicationState 集合中
Remove 方法	从 HttpApplicationState 集合中移除命名对象
RemoveAt 方法	按索引从集合中移除一个 HttpApplicationState 对象
RemoveAll 方法	从 HttpApplicationState 集合中移除所有对象
Clear 方法	从 HttpApplicationState 集合中移除所有对象
Get 方法	通过名称或者索引获取 HttpApplicationState 对象
GetKey 方法	通过索引获取 HttpApplicationState 对象名
Lock 方法	锁定对 HttpApplicationState 变量的访问以促进访问同步
UnLock 方法	取消锁定对 HttpApplicationState 变量的访问以促进访问同步
Set 方法	更新 HttpApplicationState 集合中的对象值

Application 对象还提供了索引器，可以直接像访问集合一样访问 Application 对象里存储的值。格式如下：

```
Application["keyName"] = objectValue;
Application[0] = objectValue;
Object value = Application["keyName"];
Object value = Application[0];
```

3.6.2　实例描述

上次给一个公司做了一个小的企业网站。很郁闷，那个老板真特别。一个小企业网站，非得要在页面底部显示当前在线的人数。八百年没有访问一次的网站，天天挂个数字“0”，他真也不嫌别扭。

还好，实现起来非常简单，就没说什么只管添上吧。

这个实例中就简单地演示一下 ASP.NET 统计在线人数的方法。

3.6.3　实例应用

【例 3-15】统计网站在线人数。

(1)　新建一个 Web 项目。

(2)　添加一个“全局应用程序类”，如图 3-26 所示。

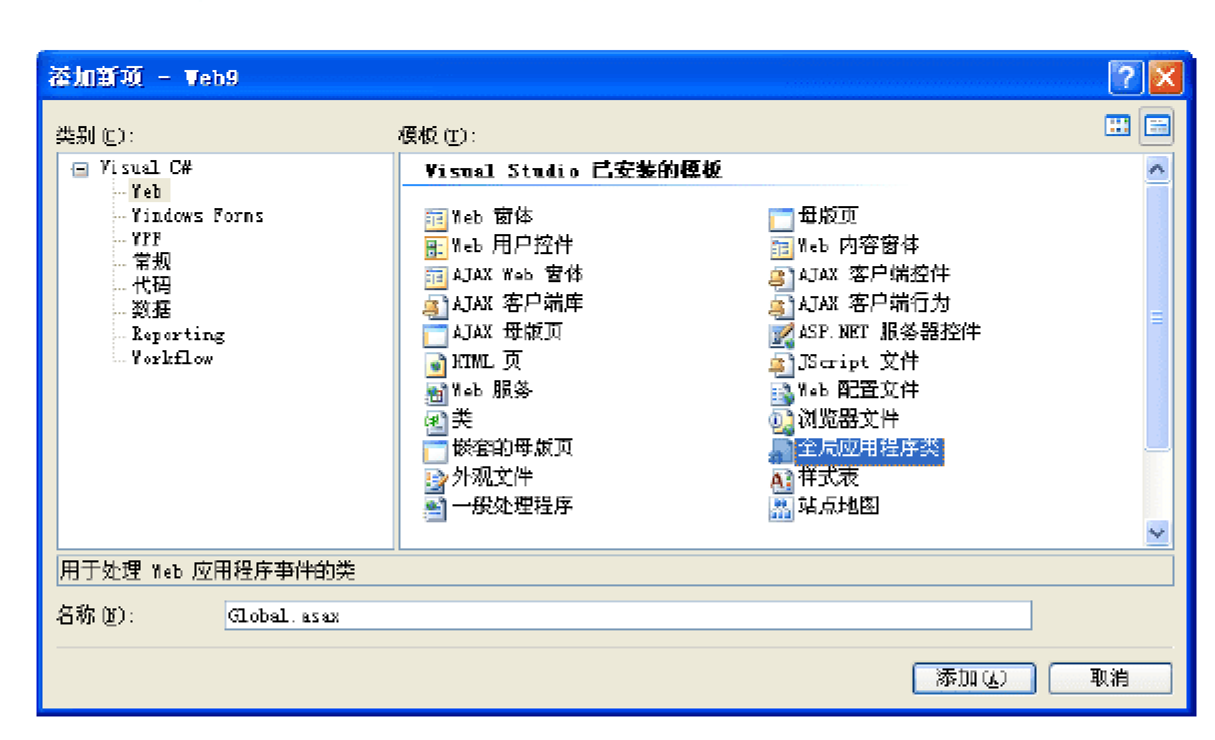


图 3-26　“添加新项”对话框

(3)　首先我们要在应用程序启动的时候初始化统计在线人数的全局参数。修改 Application_Start 方法，代码如下：

```
protected void Application_Start(object sender, EventArgs e)
{
    /* 如果“在线人数”为空，将其初始化为 0 */
    if (Application["CountOnline"] == null)
    {
        Application["CountOnline"] = 0;
    }
}
```

(4) 在创建会话对象的时候我们还要对统计在线人数的全局参数加 1。修改 Session_Start 方法，代码如下：

```
protected void Session_Start(object sender, EventArgs e)
{
    /* 为方便测试，这里设置 Session 对象的生存时间为 1 分钟 */
    Session.Timeout = 1;

    int countOnline = (int)Application["CountOnline"];    //获得在线人数
    countOnline++;    //执行累加
    Application["CountOnline"] = countOnline;  //设置当前在线人数
}
```

(5) 在会话对象销毁的时候，我们还要对统计在线人数的全局参数减 1。修改 Session_End 方法，代码如下：

```
protected void Session_End(object sender, EventArgs e)
{
    int countOnline = (int)Application["CountOnline"];    //获得在线人数
    countOnline--;    //执行递减
    Application["CountOnline"] = countOnline;  //设置当前在线人数
}
```

(6) 在 Default.aspx 页面加入如下一行代码：

```
<div>当前在线人数为：<%=Application["CountOnline"] %></div>
```

这行代码在页面上打印当前在线人数。

3.6.4 运行结果

运行项目，访问 Default.aspx 页面，访问结果如图 3-27 所示。

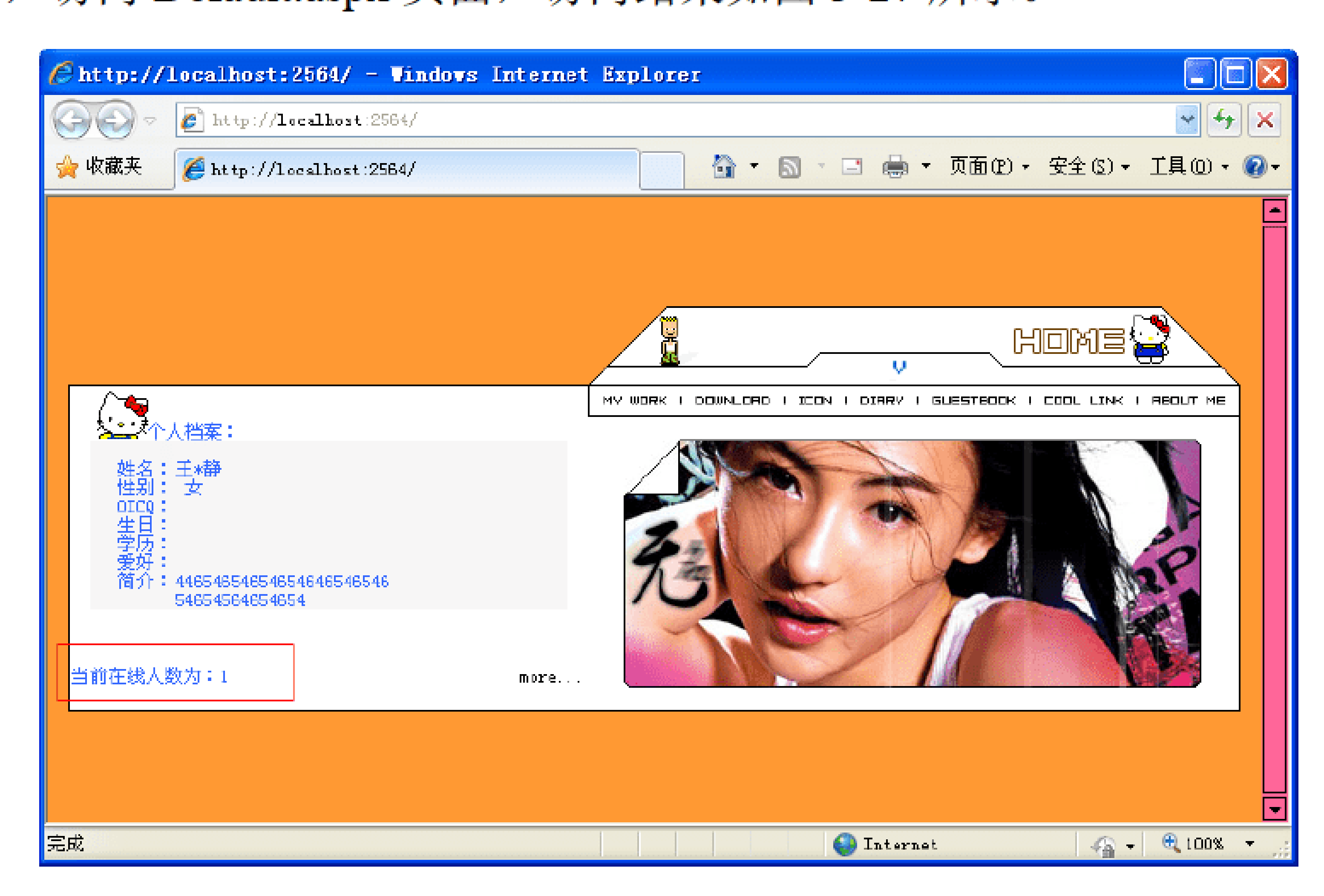


图 3-27 统计当前在线人数

共有一个浏览器访问，当前在线人数为 1。

再次打开一个 Web 浏览器，访问该页面，页面上显示的当前在线人数就会变成 2。关闭掉一个浏览器，等一分钟后，再次刷新另一个浏览器，当前在线人数就变成 1 了。



虽然我们设置 Session 对象生存周期是 1 分钟，但是到 1 分钟的时候.NET Framework 不一定会对该对象进行垃圾回收，也就不会触发对象的销毁事件，也就不会准确地在 1 分钟的时候更新当前在线人数。

3.6.5　实例分析



源码解析：

该实例在“全局应用程序类 Global”中控制整个应用程序运行状态，在 Application_Start 方法里初始化在线人数参数 Application[“CountOnline”]，在 Session_Start 方法里累加在线人数，在 Session_End 方法里递减在线人数，最后在页面上打印出来。

3.7　记录用户登录状态

Web 请求是一种无状态的请求。客户端在请求 Web 服务器以后，Web 服务器对客户端的请求做出响应，然后就互不认识了。下次访问又是一次全新的请求、响应。

但客户端往往不会只对一个 Web 服务器请求一次，所以双方难免会需要对双方的请求、响应状态进行保存。

前面我们提到过，服务器会对每个用户创建一个 Session 对象，用于保存当前用户的状态信息。下面我们来了解一下。



视频教学：光盘/videos/3/SessionObject.avi　　　　长度：5 分钟

3.7.1　基础知识——Session对象

浏览器请求 Web 服务器，Web 服务器在需要保存用户状态的时候，会为该浏览器创建一个会话状态保存对象 Session。该对象有一个唯一标识符 SessionID。在对浏览器请求做出响应的时候顺便把 SessionID 也一并发送给浏览器，浏览器在下一次访问 Web 服务器的时候将 SessionID 一并提交给服务器，服务器根据 SessionID 在服务器上查找相对应的会话状态，持续使用。

Session 对象默认保存在服务器内存中，因为 Web 方式的应用程序的访问不稳定性，为了节省开销，Session 对象都有生命周期属性。生命周期指定该对象在服务器上最大的空闲时间。如果某个 Session 对象闲置超过这个时间，就自动将该对象遗弃，随时等待垃圾回收器的回收。

Session 对象可以像数据字典一样存储和读取数据。格式如下：

```
Session["key"] = objectValue;
Session[0] = objectVlaue;
Object value = Session["key"];
Object value = Session[0];
```

Session 对象是 System.Web.SessionState.HttpSessionState 类的实例。该对象常用的属性和方法如表 3-12 所示。

图 3-12　Session对象的常用属性和方法

名　称	说　明
Contents 属性	获取对当前会话状态对象的引用
CookieMode 属性	获取一个值，该值指示是否为无 Cookie 会话配置应用程序
Count 属性	获取会话状态集合中的项数
Keys 属性	获取存储在会话状态集合中所有值的键的集合
Mode 属性	获取当前会话状态模式
SessionID 属性	获取会话的唯一标识符
Timeout 属性	获取并设置在会话状态提供程序终止会话之前各请求之间所允许的时间(以分钟为单位)
Abandon 方法	取消当前会话
Add 方法	向会话状态集合添加一个新项
Clear 方法	从会话状态集合中移除所有的键和值
Remove 方法	删除会话状态集合中的项
RemoveAll 方法	从会话状态集合中移除所有的键和值
RemoveAt 方法	删除会话状态集合中指定索引处的项

3.7.2　实例描述

在一般的管理系统中，需要在管理界面进行用户权限的验证，所以这需要让用户登录并且使用户状态保持。

用户登录过后，会在系统管理界面展示一些当前登录用户的个人信息，比如账号等。

本实例中我们就在用户登录成功以后在管理主界面上展示一下用户的登录账号。

3.7.3　实例应用

【例 3-16】记录用户的登录状态。

(1)　新建一个 Web 项目。

(2)　添加一个登录页面 Login.aspx。

(3)　修改 Login.aspx 的页面文件，添加一个“用户名”文本框、一个“密码”文本框，以及一个“登录”按钮。主要代码如下：

```
<asp:TextBox ID="txtUsername" CssClass="input" runat="server"></asp:TextBox>
<asp:TextBox ID="txtPassword" CssClass="input" TextMode="Password"
  runat="server"></asp:TextBox>
<asp:Button ID="btnLogin" runat="server" Text="Login" onclick="btnLogin_Click" />
```

(4) 添加“登录”按钮的单击事件，在单击事件处理程序中进行登录验证。这里只简单地测试如果用户名不为空就算成功登录。代码如下：

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    string username = this.txtUsername.Text.Trim();    //获取用户名

    if (username != "")
    {
        /* 在会话中记录用户登录状态 */
        Session["CurrentUser"] = this.txtUsername.Text;
        Response.Redirect("Default.aspx");    //跳转到管理主界面
    }
    else
    {
        Response.Write("<script>alert('请输入用户名和密码！');</script>");
    }
}
```

(5) 在用户管理主页面 Default.aspx 中，打印登录用户的账号，在页面的适当位置添加如下一行代码：

```
超级管理员：<%= Session["CurrentUser"] %>
```

这里使用页面小脚本直接输出 Session 中的当前用户的信息(即用户登录名)。

3.7.4　运行结果

运行项目。访问 Login.aspx 页面，效果如图 3-28 所示。



图 3-28　用户登录界面

在登录界面中我们输入用户名和密码(这里用户名输入的是 admin)。然后单击 Login 按钮。登录成功，跳转到管理系统首页，如图 3-29 所示。

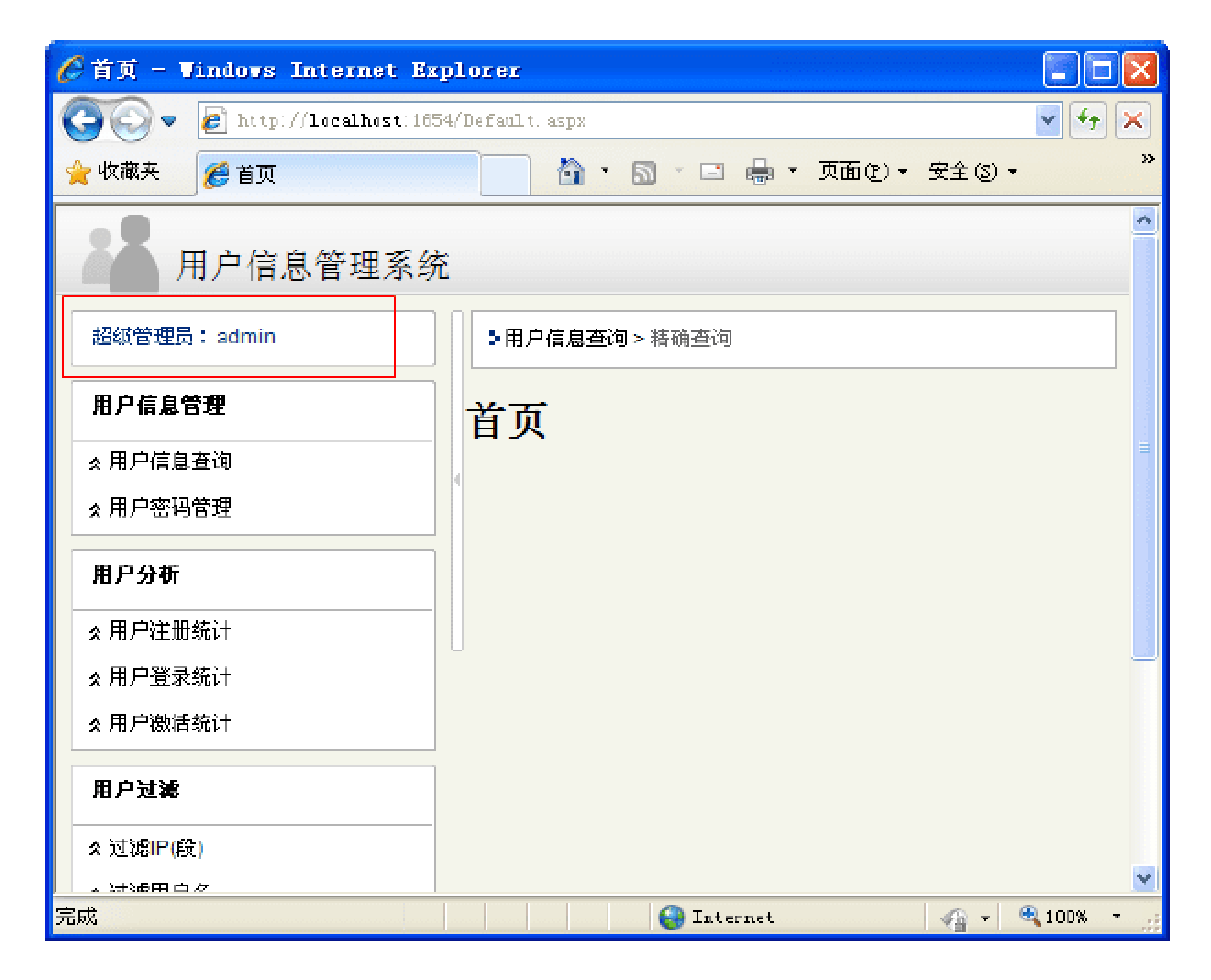


图 3-29　管理系统首页

3.7.5　实例分析


源码解析：

该实例在用户登录的时候进行用户验证，如果验证成功，将用户信息使用 Session 对象的索引器保存到会话对象 Session 中，然后使浏览器跳转到用户管理主界面，并在管理主界面中使用 Session 对象的索引器访问得到前面保存的用户信息，在页面中展示出来。

另外，我们还可以根据该 Session 对象是否保存用户登录信息来判断用户是否登录。

3.8　缓存页面的信息

前面我们讲过，可以使用页面@OutputCache 指令来缓存整个页面的内容。

但是很多时候我们只想暂时保存某一些数据，并不想让整个页面都缓存起来，所以就不能再使用页面@OutputCache 指令了。

当然，其实我们还可以将其保存到全局应用程序对象 Application 中。不过在 Application 中的值将在程序运行时一直存在，而我们只需要缓存一段时间，所以使用 Application 对象的话就比较费事了。

ASP.NET 提供了一个 Cache 对象来执行对对象数据的缓存，下面来研究一下。


视频教学：光盘/videos/3/CacheObject.avi　　　　长度：5 分钟

3.8.1　基础知识——Cache对象

Cache 对象是 System.Web.Caching.Cache 类的一个实例，并且只要对应的应用程序域保持活动，该实例便保持有效。该对象被作为一个只读属性声明在 HttpContext 对象或 Page 对象内。

Cache 类提供了一些方法和属性来执行对象的缓存操作，如表 3-13 所示。

表 3-13　Cache类的方法和属性

名　称	说　明
Count 属性	获取存储在缓存中的项数
Add 方法	将指定项添加到 Cache 对象，该对象具有依赖项、过期和优先级策略以及一个委托
Get 方法	从 Cache 对象检索指定项
Insert 方法	向 Cache 对象插入项
Remove 方法	从应用程序的 Cache 对象移除指定项

Cache 对象的 Insert 方法可以向应用程序缓存中添加项。通过该方法的重载，可以添加各类不同的选项，以设置依赖项、过期和移除通知。还可以使用 Add 方法向缓存添加项，它可以设置与 Insert 方法相同的所有选项，不同的是 Add 方法将返回添加到缓存中的对象。



使用 Insert 方法向缓存添加项时，如果已经存在与现有项同名的项，则缓存中的现有项将被替换。但如果使用 Add 方法，并且缓存中已经存在与现有项同名的项，则该方法不会替换该项，并且不会引发异常。

使用 Insert 方法向 Cache 中添加项，其具体代码如下：

```
Cache.Insert("keyName", objectValue);
```

通过使用 Add 方法向 Cache 中添加项，其具体代码如下：

```
Cache.Add("keyName",
    objectValue, null,
    System.Web.Caching.Cache.NoAbsoluteExpiration,
    System.Web.Caching.Cache.NoSlidingExpiration,
    System.Web.Caching.CacheItemPriority.Default,
    null);
```

使用 Get 方法获取 Cache 中的对象，其具体代码如下：

```
Object value = Cache.Get("keyName");
```

使用 Remove 方法移除项目，具体代码如下：

```
Cache.Remove("keyName");
```

另外，我们还可以使用 Cache 对象的索引器对其进行操作，代码如下：

```
Cache["keyName"] = objectValue;
Object value = Cache["keyName"];
Cache["keyName"] = null;
```

3.8.2 实例描述

我们在处理一些常用但没必要绝对及时的数据时，往往会用到数据缓存。比如 CSDN 论坛的帖子列表，就是缓存了几秒钟的。

缓存几秒钟对于小的网站可能并没有什么大不了的，不过对于这种大型的门户系统，每一秒钟就可能有成百上千(甚至更多)次访问，对这些常用的数据进行缓存，哪怕是一秒钟，就可能由数百次请求数据库缩减为一次。用户体验的变化微乎其微，但对于数据库的压力，一下子减少了上百倍。

接下来我们将某一时刻的时间作为目录数据缓存 10 秒钟来演示一下 Cache 的用法。

3.8.3 实例应用

【例 3-17】缓存页面的信息。

(1) 新建一个 Web 项目。

(2) 在 Default.aspx 页面的视图界面添加一个 Label 标签。代码如下：

```
<asp:Label ID="lblValue" runat="server"></asp:Label>
```

(3) 修改页面后台代码的 Page_Load 方法，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    DateTime time = DateTime.Now;
    if (Cache["timeKey"] == null)
    {
        /* 将时间对象插入到 Cache 中，设置到期时间为 10 秒钟后，到期后 0 秒销毁保存的值 */
        Cache.Insert(
          "timeKey", time, null, DateTime.Now.AddSeconds(10), TimeSpan.Zero);
    }
    else
    {
        time = DateTime.Parse(Cache["timeKey"].ToString());
    }
    this.lblValue.Text = time.ToString();
}
```

3.8.4 运行结果

运行项目，访问 Default.aspx 页面。

访问结果如图 3-30 所示。

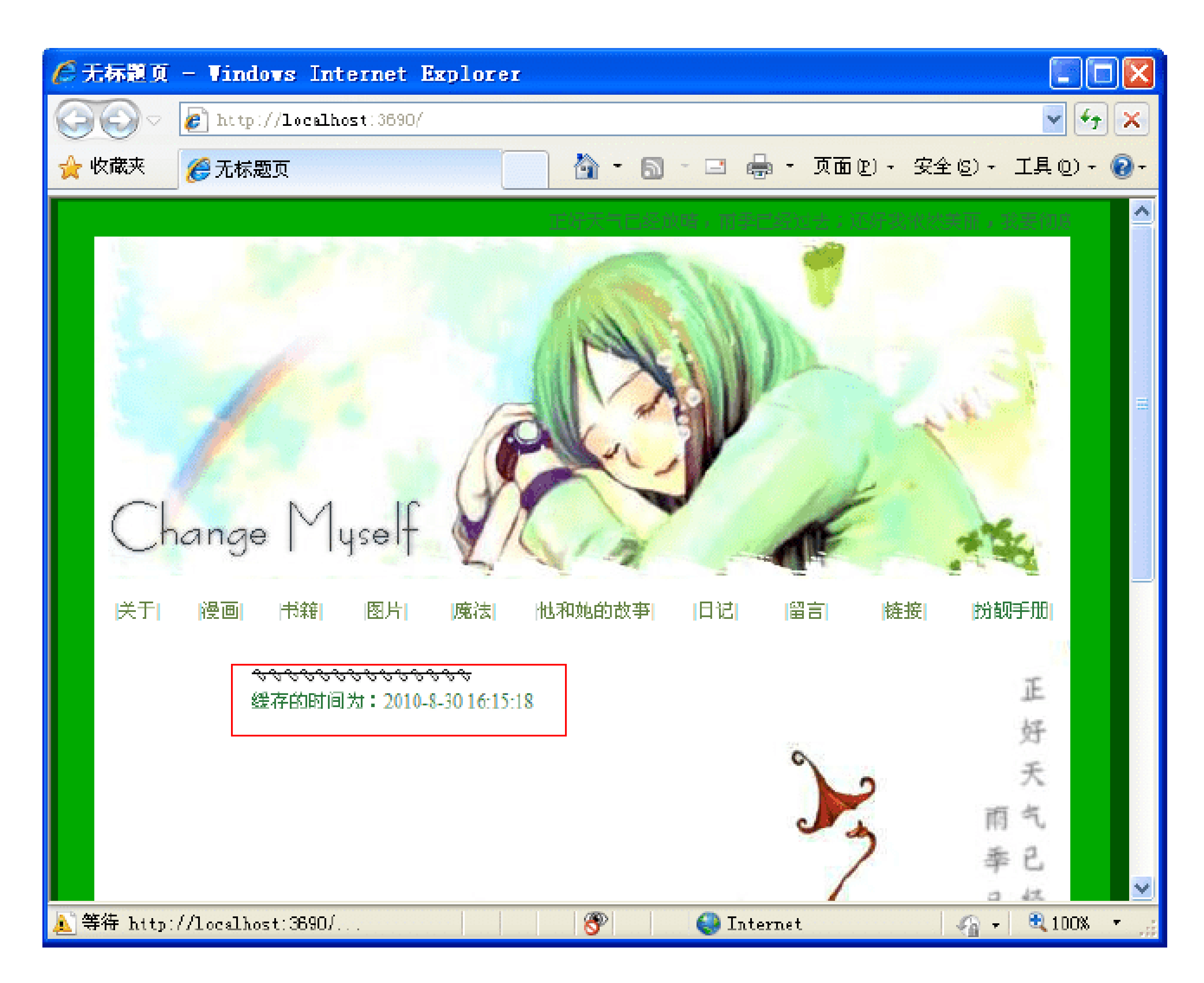


图 3-30　缓存时间

在我们第一次访问过以后的 10 秒钟内，无论是刷新页面还是在新浏览器里访问该页面，时间都不会改变。10 秒钟后，缓存的内容销毁，重新缓存数据。

3.8.5　实例分析



源码解析：

实例先获取当前时间，再检查缓存的数据是否为空，如果为空，说明还没有缓存数据，此时对数据进行缓存；如果不为空，说明缓存的数据还没有到期，则取出缓存的数据使用。

因为缓存里的数据不稳定，所以在使用缓存的时候需要检查缓存的数据是否存在。

3.9　页 面 对 象

在 ASP.NET 中，任何事物都是对象，包括 Web 页面。

ASP.NET 会把每一次请求的页面封装为一个 Page 对象，用来接收并处理用户的请求。既然是对象，肯定保存在服务器内存里。既然在服务器内存里，那么它与浏览器请求是什么关系呢？WebForm 机制的 ASP.NET 窗体是怎么实现的呢？

接下来就我们来了解一下。

3.9.1　页面的生命周期

我们知道，HTTP 是一种无状态的协议，在它下面运行的 Web 服务也不可能保存运行状态，那么 ASP.NET 的 Web 窗体如何做到准确无误地进行控件操作呢？

答案就是 ViewState。

ViewState 对象也是一个作为 Page 对象的属性的内置对象，它保存的是当前页的状态信息。

所有的页面控件的状态都被以某种形式进行编码，然后保存到 ViewState 对象中。最后将 ViewState 对象经过某种形式的编码，生成一个长而复杂的字符串，并将该字符串保存到一个 HTML 隐藏域中，一并发送给客户端。

所以，我们在客户端查看源文件的时候往往会在 body 里面看到如下一段代码：

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKMjAxNjMxMDg1Nw9kFgICAw9kFgICFA8QZGQWAGQYAJDAFDFJhZGlvQnV0dG9
uMQUMUmFkaW9CdXR0b24xxmIt/JmAIinSc30PejMngC2rR38=" />
```

这里面就是 ViewState 对象保存的服务器控件状态。

在用户对页面的控件进行触发服务器事件的操作时，页面表单会自动提交。服务器在接收到用户的提交时会去分析提交过去的 ViewState 对象的状态(就是上面那堆字符)，分析完后得到页面服务器控件的状态，重新设置每一个控件状态，然后执行用户所触发的控件事件。最后执行完事件处理程序，生成执行结果，返回给客户端。

这里在返回的时候仍然会重新保存页面控件状态，生成 ViewState 对象，发送给页面的隐藏域。

整个执行过程如图 3-31 所示。

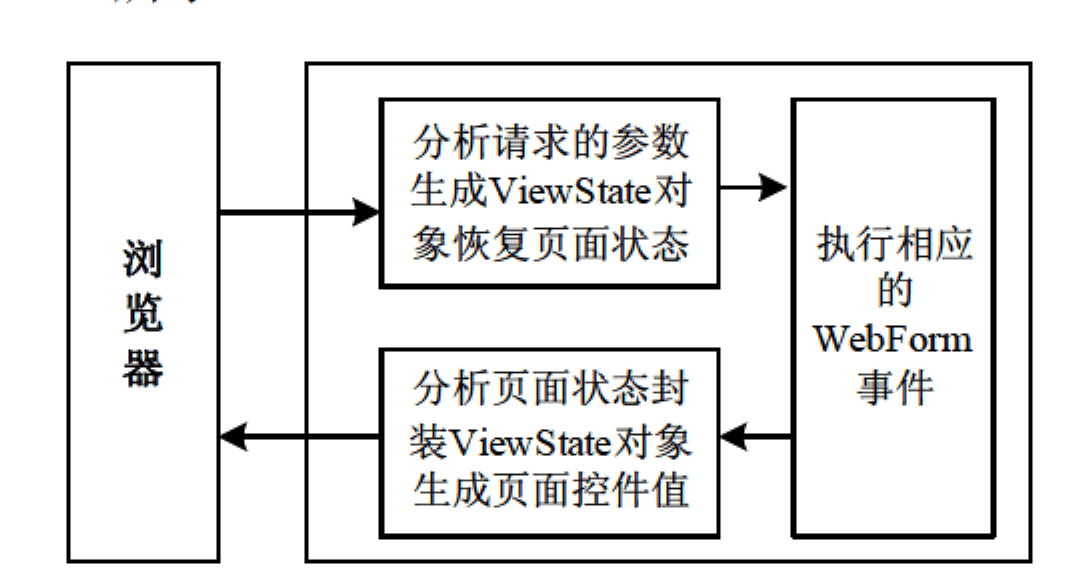


图 3-31　ASP.NET Web窗体状态保持工作流程

刚才我们已经知道，每一次请求都需要对页面控件的状态进行初始化和恢复。所以我们大概可以知道，页面 Page 对象每次都需要重建(否则就不用每次都恢复状态了)。

要了解 Page 对象的生存周期，我们可以根据 Page 对象的几个内置的事件来研究一下，如表 3-14 所示。

表 3-14　Page对象的事件

事　件	说　明
PreInit	在页面初始化开始时发生
InitComplete	在页面初始化完成时发生
PreLoad	在页面内容加载之前发生
LoadComplete	在页面生命周期的加载阶段结束时发生
PreRenderComplete	在呈现页面内容之前发生
SaveStateComplete	在页面已完成对页和页上控件的所有视图状态和控件状态信息的保存后发生

从上表中我们可以知道，Page 对象执行了初始化、加载、呈现内容、保存状态等几个阶段。不过微软件官方将 Page 对象的页面生命周期做了更加详细的划分，更能说明问题。

微软官方将 Page 对象的生存周期分为 10 个阶段，如图 3-32 所示。

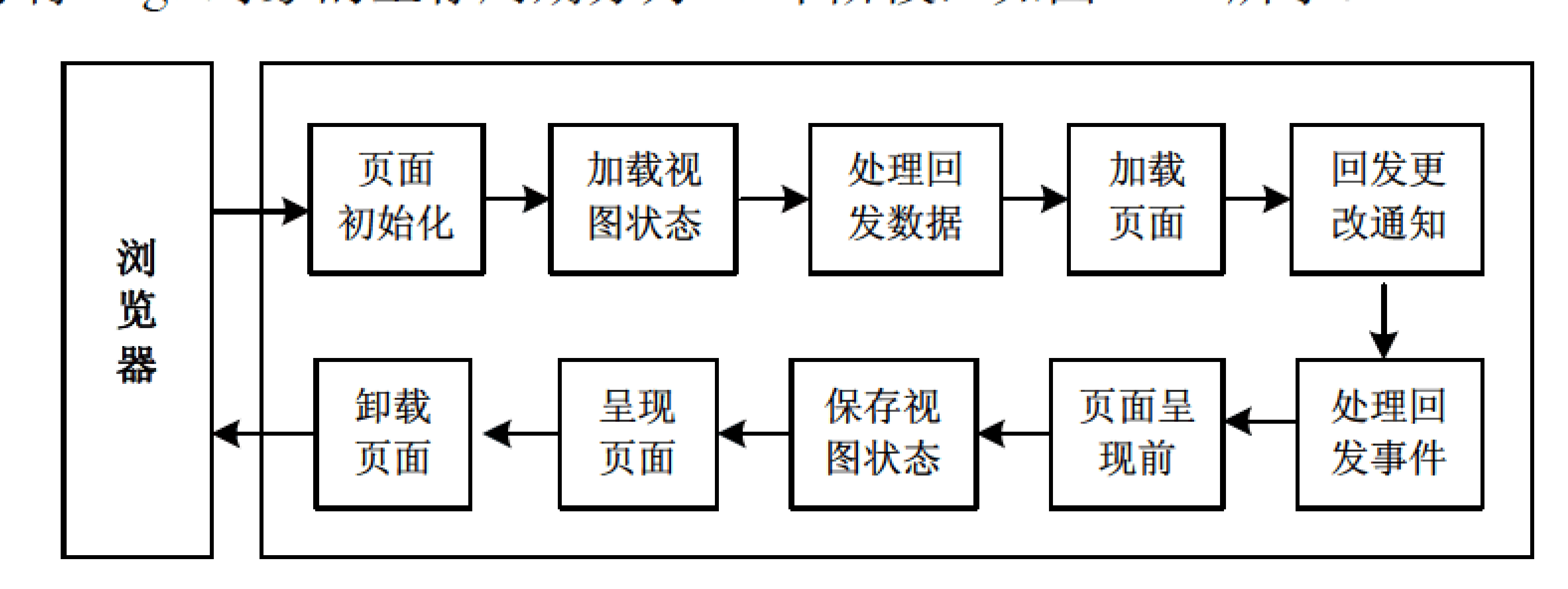


图 3-32　Page对象的生存周期

(1) ASP.NET 接收到用户请求以后创建相应的页面对象。同时开始初始化页面控件，设置页面控件的默认状态，默认值，绑定各个事件处理程序。

(2) 初始化页面默认状态以后，就立即开始加载页面视图状态。默认情况下，视图状态存储在一个页面隐藏控件中，控件的名字是__VIEWSTATE，是 ASP.NET 默认自动加载到页面里的。ASP.NET 获取__VIEWSTATE 的值以后分析并修改视图中各控件的默认状态，使其与上次的请求保持一致。

(3) 加载完页面的视图状态，下一步就开始处理回发的数据。处理回发数据阶段使各个控件根据请求来的数据更新视图控件的状态。比如 Checkbox 控件更新 Checked 状态，TextBox 控件更新其文本内容，DropDownList 控件选中用户最新的选择操作。

(4) 处理回发数据阶段结束时，页面已根据客户端修改的值更新页面控件原来的值。页面开始激发 Load 事件。

(5) 这时如果页面控件触发了某些已声明的事件，比如修改 TextBox 的值触发了 TextChanged 事件等。

(6) 得到这些事件触发通知以后，在页面 Load 事件以后开始触发相应的控件回发事件。

(7) 处理完控件的回发事件，页面就开始做呈现前的准备，进行最后的修改状态操作(因为在回发事件中可能修改了控件状态)。

(8) 接着，页面进入保存视图状态操作。所有的控件在这里刷新自己在 ViewState 里的状态信息，所得到的 ViewState 对象最终经过序列化等一些操作关联到页面__VIEWSTATE 隐藏字段中。

(9) 然后，ASP.NET 根据上面的操作生成相应的视图代码：HTML 页面，并发送到请求响应流中。

(10) 最后，页面触发卸载事件，释放相应资源。

此事件之后，客户端浏览器将收到 Web 服务器的 HTTP 响应数据，处理并呈现在页面上。

讲完了页面生存周期，最后我们看一下 Page 对象。

Page 对象是 System.Web.UI.Page 类的实例，作为公有属性被声明在 System.Web.UI.Control 类中，该类又被 System.Web.UI.TemplateControl 类继承，Page 类又继承自 TemplateControl 类。所以在页面中可以直接访问公有对象 Page。

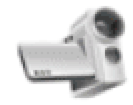
Page 包含上面我们讲过的那些对象，比如 Server、Request、Response、Application、Session、ViewState、Cache 等。这些对象都是 Page 对象的属性，可以在 Page 类内部直接访问。

3.9.2 判断回调时机

在 ASP.NET 中，我们经常在页面加载的时候编写代码对页面上的控件进行初始化。但是，页面加载是在处理回发数据之后。所以如果我们在页面加载的时候对页面进行初始化了，那么无论怎么对页面进行操作，最终处理的时候都将是初始化的值。

所以我们需要在页面第一次加载的时候对页面进行初始化，以后在页面回发的时候不再初始化。

在这里 ASP.NET 提供了一个 IsPostBack 属性进行标识，使我们很方便地解决了这个问题。


视频教学：光盘/videos/3/pageCommand.avi　　**长度：**11 分钟

1. 基础知识——IsPostBack属性

第一次访问一个页面时，一般都是直接在地址栏请求该页面的 URL，所以这种方式是执行 GET 方式的请求。而在触发页面控件的事件时一般是以 POST 方式提交的请求。所以，我们用直接获取当次请求的方式就可以确定当次请求用不用对页面进行初始化。

Page 类提供了一个 IsPostBack 属性，该属性是一个布尔型的公有只读属性，该属性值指示该页面是否为响应客户端回发而加载。

直白一点地说，就是如果该次请求是 POST 方式，属性将返回 true，否则将返回 false。

2. 实例描述

这个属性在实际应用中经常用到。

比如我们写了一个简单的算术计算器，初始时要对页面的文本框进行初始化。我们就需要使用到它。

本实例我们就用一个简单的加法计算器来演示一下它的用法。

3. 实例应用

【例 3-18】判断回调机制。

(1) 新建一个 Web 项目。

(2) 在 Default.aspx 页面的视图界面拖入两个文本框，ID 分别为 txtNum1 和 txtNum2，用于接收用户输入的两个值。添加一个命令按钮，ID 属性为 btnAdd，Text 属性为“加法”，用于接收用提交命令。另外再添加一个标签控件，ID 属性为 lblResult，用于展示运算结果。

页面的主要代码如下：

```
<br />Num1: <asp:TextBox ID="txtNum1" runat="server"></asp:TextBox>
<br />Num2: <asp:TextBox ID="txtNum2" runat="server"></asp:TextBox>
<br />
<asp:Button ID="btnAdd" runat="server" Text="加法" onclick="btnAdd_Click" />
<br />执行结果为：<asp:Label ID="lblResult" runat="server"></asp:Label>
```

(3) 添加命令按钮 btnAdd 的单击事件，修改事件处理程序。具体代码如下：

```
protected void btnAdd_Click(object sender, EventArgs e)
{
    /* 获取页面用户输入的两个值 */
    decimal num1 = decimal.Parse(this.txtNum1.Text);
    decimal num2 = decimal.Parse(this.txtNum2.Text);
    decimal result = num1 + num2;    //计算结果
    this.lblResult.Text = num1 + " + " + num2 + " = " + (result);  //展示结果
}
```

(4) 另外不能忘了对页面进行初始化。修改 Page_Load 方法，具体代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)    //如果页面是第一次请求
    {
        this.txtNum1.Text = "0";
        this.txtNum2.Text = "0";
    }
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面，如图 3-33 所示。

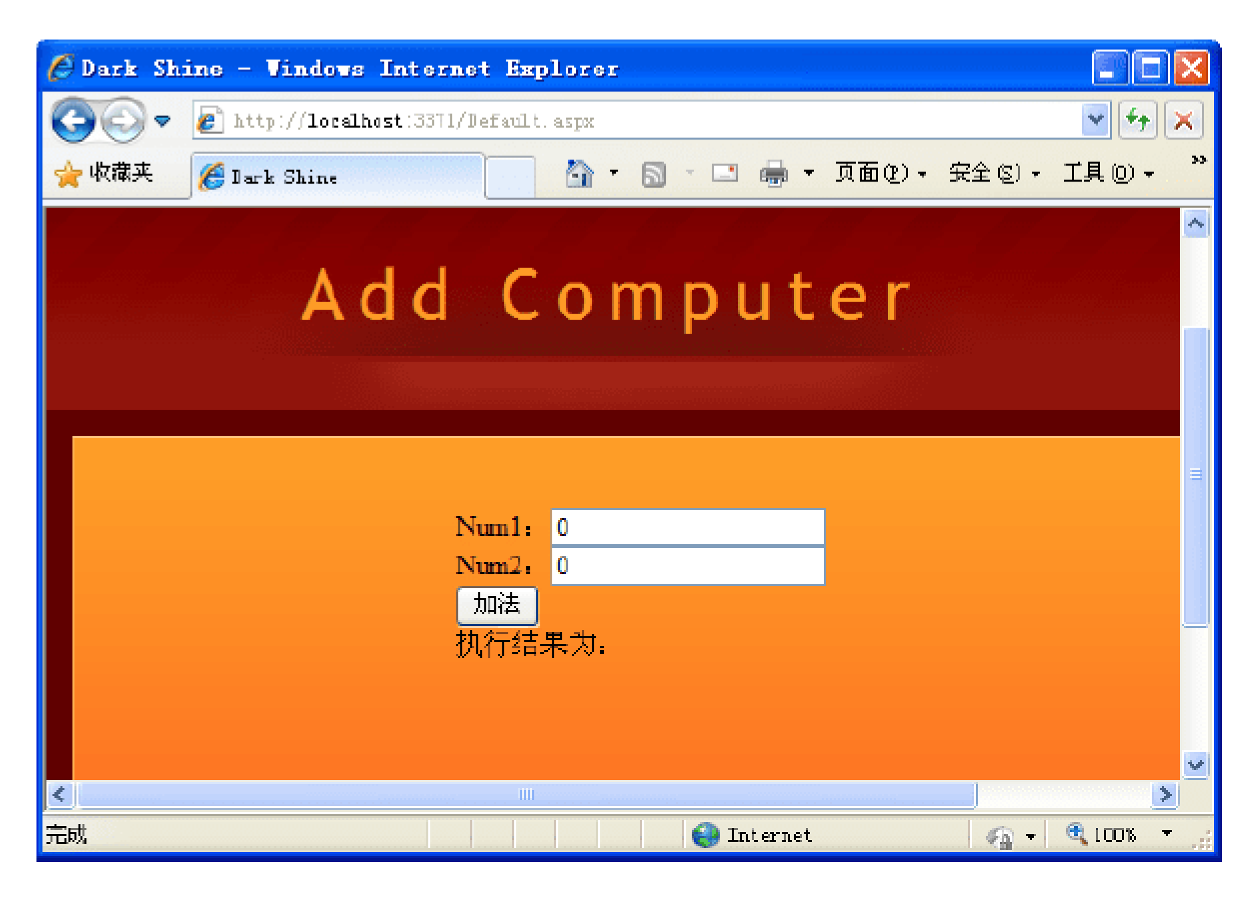


图 3-33　加法计算器

在 Num1 文本框里输入数字 20.2，在 Num2 文本框中输入数字 10，单击“加法”按钮，页面即可计算出加法执行的结果，如图 3-34 所示。

5. 实例分析



源码解析：

实例使用 Page 对象的 IsPostBack 属性得到请求方式，在不是以 POST 方式请求页面的时候，对文本框的值进行初始化。

这样在第一次访问页面的时候，文本框里默认都会有数值，而在单击“加法”按钮的时候不会再执行初始化操作，不会影响加法计算操作的正确执行。

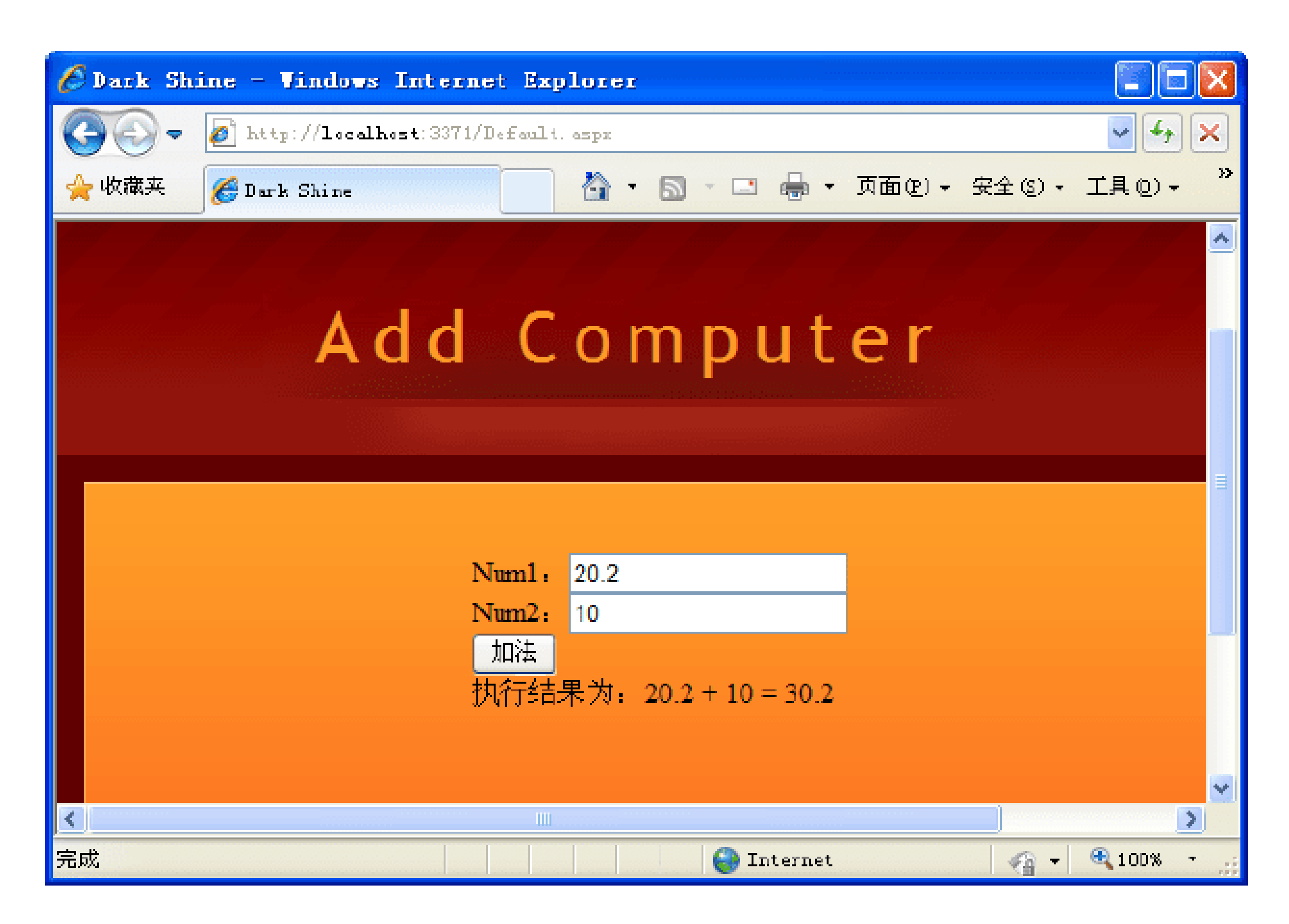


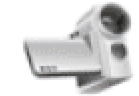
图 3-34　执行加法运算

3.9.3　输出客户端脚本

在以前的实例中，我们使用了 Response 对象的 Write 方法直接向页面打印了一行 HTML 标签，来弹出一个对话框。

这样使用起来比较方便，但是使用多了就会发现：有时候在一些 DIV+CSS 布局的 Web 页面中，使用上面的方法向面页打印 HTML 文本以后，页面布局就会因此而乱掉，而且在弹出对话框的时候浏览器一片空白，给人感觉很不好。

Page 对象还提供了一个 ClientScript 属性，该属性可以向页面表单内部注册一些脚本。

 **视频教学：** 光盘/videos/3/pageCommand.avi　　长度：11 分钟

1. 基础知识——Page对象

Page 对象的 ClientScript 属性是一个 System.Web.UI.ClientScriptManager 类的实例。该类提供了在 Web 应用程序中管理客户端脚本的方法。

其中常用的向页面注册脚本的方法如表 3-15 所示。

表 3-15　ClientScriptManager类的常用方法

方 法 名	说　明
RegisterArrayDeclaration	向页面对象中注册 JavaScript 数组声明
RegisterClientScriptBlock	向页面对象中注册客户端脚本
RegisterClientScriptInclude	向页面对象中注册引入脚本文件的客户端脚本
RegisterClientScriptResource	向页面对象中注册客户端脚本资源
RegisterExpandoAttribute	对指定控件注册自定义属性和值

续表

方 法 名	说　明
RegisterForEventValidation	为验证事件注册引用
RegisterHiddenField	给指定对象注册一个隐藏值
RegisterOnSubmitStatement	向页面对象注册 OnSubmit 语句，该语句在客户端提交表单时执行
RegisterStartupScript	向页面对象注册启动脚本

如同 Response 对象的 Write 方法一样，向页面注册脚本的方法有两个，分别是 RegisterClientScriptBlock 和 RegisterStartupScript。

RegisterClientScriptBlock 方法向页面 form 表单的开始标签后面添加一行脚本，而 RegisterStartupScript 方法向页面 form 表单的结束标签前面添加一行脚本。

2. 实例描述

本实例我们使用 Page 对象的 ClientScript 属性的 RegisterStartupScript 方法重构一下向页面注册脚本的功能。

3. 实例应用

【例 3-19】输出客户端脚本。

(1) 新建一个 Web 项目。

(2) 在 Default.aspx 页面添加登录功能使用的控件(文本框和命令按钮)。

(3) 修改“登录”按钮的单击事件处理程序，具体代码如下：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    if (this.txtUsername.Text != "admin")
    {
        /* 向客户端注册脚本 */
        ClientScriptManager csm = Page.ClientScript;
        csm.RegisterStartupScript(GetType(),
            "", "<script>alert('用户名不存在！');</script>");
    }
}
```

4. 运行结果

运行项目，访问 Default.aspx 页面，如图 3-35 所示。

输入用户名 administrator(只要不是 admin 即可)，密码随意，单击“登录”按钮，结果如图 3-36 所示。

5. 实例分析



源码解析：

本实例中，用户登录的时候验证用户登录账号，在用户输入的账号不等于指定账号时，向页面注册提示脚本。

向页面注册脚本先使用 Page 对象的 ClientScript 属性获取当前客户端脚本管理对象(ClientScriptManager 类的实例)，并使用该客户端脚本管理对象的 RegisterStartupScript 方法向当前页面注册一段 JavaScript 代码。



图 3-35 登录页面



图 3-36 运行结果

3.10 常见问题解答

3.10.1 .aspx如何绑定一个.cs文件



.aspx 如何绑定一个.cs 文件？

网络课堂：http://bbs.itzcn.com/thread-9854-1-1.html

复制了 a.aspx 和 a.cs 两个文件，可是复制出来的“a.aspx”绑定的代码依旧是 a.cs，如何

绑定到新的那个.cs？

【解决办法】：嗯……特简单，只需将.aspx 文件中 Page 指令的相关值改掉，例如：

```
<%@ Page language="C#" Codebehind="a.aspx.cs" AutoEventWireup="false"
  Inherits="a" %>
```

将 Codebehind 值改成你要绑定的.cs 文件，将 Inherits 改成.cs 文件中 Page 子类的名称。注意，如果你要绑定的.cs 文件没有添加至 project，就要通过“添加已有项”添加至 project。

3.10.2　如何在页面中使用用户控件



如何在页面中使用用户控件？
网络课堂：http://bbs.itzcn.com/thread-9855-1-1.html

如何在.aspx 页面中使用用户控件(用户自定义的一个控件)？

【解决办法】：嗯……最简单的办法，是从解决方案管理器中把.ascx 直接拖进.aspx 设计界面。当然，你也可以在 HTML 代码里手动添加，例如：

```
<uc1:control1id="Control" runat="server"></uc1:control1>
```

不过一定要记得先注册：

```
<%@Register TagPrefix="uc1" TagName="Control1" Src="Control1.ascx" %>
```

3.10.3　怎样让response.write输出的内容出现在body内



怎样让 response.write 输出的内容出现在 body 内？
网络课堂：http://bbs.itzcn.com/thread-2688-1-1.html

response.write 输出的内容总会出现在 HTML 代码的最前头。

怎么样才能把输出的内容放到 body 标签内，或者按钮触发事件里面。用 label.text 方法可以的，不过那样产生的 viewstate 内容太长了。

【解决办法】：先声明一个变量：

```
public string str;
```

然后在页面 HTML 代码中加入：

```
<body>
<%=str%>
</body>
```

这样和 response.write 是一样的效果。

或者用下面的语句禁用 viewstate 属性：

```
Label1.EnableViewState = false;
```

3.11 习　　题

一、填空题

(1) Request 对象的 QueryString 属性是用来获取以__________方式提交的数据。

(2) 在页面@Page 指令中设置页面自动绑定控件事件的是__________属性。

(3) 补充下面的代码，设置输出页面为 XML 文件，编码为 UTF-8:

```
Response.ContentType = "__________";
Response.Charset = "utf-8";
```

(4) 将页面请求转发到一个页面地址，地址栏 URL 保持不变。这个功能需要使用__________对象下的方法。

(5) 想要直接在页面打印一些文本信息，需要用到 Response 对象的__________方法。

(6) 英文字符空格“ ”在经过 HTML 编码以后的代码为__________。

二、选择题

(1) 在服务器端保存登录用户的状态，需要使用 ASP.NET 提供的__________对象。

A. Request　　B. Server　　C. Session　　D. Cache

(2) Response 对象的 Write 方法有__________个重载。

A. 2　　B. 3　　C. 4　　D. 5

(3) 下面__________方法可以对 HTML 编码过的文本进行解码。

A. UrlEncode　　B. UrlDecode　　C. HtmlEncode　　D. HtmlDecode

(4) 缓存某个数据对象可以__________。

A. 使用页面@OutputCache 指令　　B. 使用 Application 对象

C. 使用 Cookie 对象　　D. 使用 Cache 对象

(5) 在使用 Session 对象的 Timeout 属性设置该对象生存周期的时候，设置的数值是以__________为单位的。

A. 小时　　B. 分钟　　C. 秒　　D. 毫秒

三、上机练习

上机练习：用户登录。

所有的管理系统都需要用户登录的功能。

这次练习要求实现一个完整的用户登录功能。

用户在登录页面输入用户名和密码，验证登录成功以后跳转到系统管理界面，并在页面给用户展示用户登录的账户名以及用户登录 IP。

如果用户没有登录而直接访问管理页面，则将用户请求跳转到登录页面。

登录页面如图 3-37 所示。

登录成功时进入管理页面，如图 3-38 所示。



图 3-37　登录页面



图 3-38　登录成功



第 4 章　ASP.NET的拿来主义

内容摘要：

在开发项目的过程中，ASP.NET 为开发人员提供了一个方便、快捷的开发平台。它提供了许多可利用的资源，比如说数组、集合列表、日期和时间、处理字符串的一些常用方法、泛型等。

那么所谓的“拿来主义”，就是为了满足项目的需求，在开发项目的过程中，可以使用 ASP.NET 提供的可利用资源，即开发环境封装好的一些类、结构、枚举、接口等。

在本章中，将会介绍一些在 Web 开发中常用的知识点。为了更方便、更快捷地使读者体会到 ASP.NET 的拿来主义，将用控制台应用程序来为读者展示。

学习目标：

- 了解并掌握数组的使用
- 掌握集合列表的使用
- 掌握日期和时间的使用
- 掌握字符串的常用操作
- 了解并掌握数据类型之间的转换方法
- 了解并掌握泛型的声明及使用

4.1 使 用 数 组

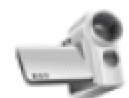
.NET Framework 提供了一种被称为“集合”的对象，使用这些对象可以处理或操作一组具有相同特性的元素。数组也是一种集合对象，它可以将多个类型相同的元素组合为一个整体，从而通过数组统一访问或操作这些元素。本节将讲解使用数组的方法，主要介绍以下知识点：

- 声明数组
- 初始化数组
- 遍历数组
- 多维数组
- 数组的排序

通过本节的学习，读者将可以了解到声明和初始化数组的基本操作方法。

4.1.1 获取一个成绩

初始化数组之后，每一个元素都有它对应的值。

 视频教学：光盘/videos/4/Array.avi　　　　长度：6 分钟

1. 基础知识——数组的声明与初始化

(1) 声明一个成绩数组

数组是有序数据的集合。数组中的每一个元素都属于同一个数据类型，用一个统一的数组名和下标来唯一确定数组中的元素。数组元素的数据类型被称为元素类型。数组的元素可以为任何类型，甚至可以是数组。



数组的下标从 0 开始。例如一个数组名为 s，s[0]就表示 s 数组的第一个元素。

在使用数组之前，必须对数组进行声明。声明数组的语法如下：

```
type []arrayName;
```

其中，type 表示数组元素的类型，如 int、double、string、object 等；arrayName 表示数组的名称。

(2) 初始化成绩数组——使用 new 操作符，并指定数组的长度

初始化数组就是为数组中的每一个元素赋值。数组实例化后，每个元素都为空，必须初始化赋值，才具有实际运算的意义。初始化数组存在多种方法，在这个小节中，我们将要讲解的是其中一种——使用 new 操作符，并指定了数组的长度。



如果数组没有被初始化，那么该数组的每一个元素的值为其元素类型的默认初始值。

下面将一组值{95,65,89,75,63}赋给 score 数组。该初始化操作使用了 new 操作符，并指定了数组的长度为 5：

```
int []score = new int[5] {95,65,89,75,63};
```

该数组初始化之后，其元素的值分别为 95、65、89、75 和 63。

(3)　获取数组元素的值

通过使用数组名称以及相应的索引来指定单个元素可以获取一个数组中的值。

①　在表达式中指定数组名，后接括号。

②　在括号内，对应于要获取的元素，为每个索引提供一个表达式。数组的每一维都需要一个索引。例如，score[2]，它表示 score 数组的索引为 2 的索引值，也就是数组的第三个元素的值。



对于数组的每一维，GetUpperBound 方法都返回索引可具有的最大值。最小索引值始终为 0。

2. 实例描述

人类是一个集合体，每一个人都有他自己特有的称呼，就是他的名字。上学的时候，每次考试过后，老师都会给学生排名，每个排名后面标示着这个学生的成绩，如果想要查看某个学生的成绩，可以根据排名来查，也可以根据名字来查，前者会快一点。

前面我们讲到了数组，一个存放数据的集合体。同样，我们也可以根据数组的索引找到它的索引值。这里我们将要为大家讲解如何获取数组中一个元素的值。

3. 实例应用

【例 4-1】获取一个成绩。

控制台应用程序代码如下：

```
int []score = new int[5] { 95, 65, 89, 75, 63 };
Console.WriteLine(score[2]);
```

4. 运行结果

运行此程序，效果如图 4-1 所示。

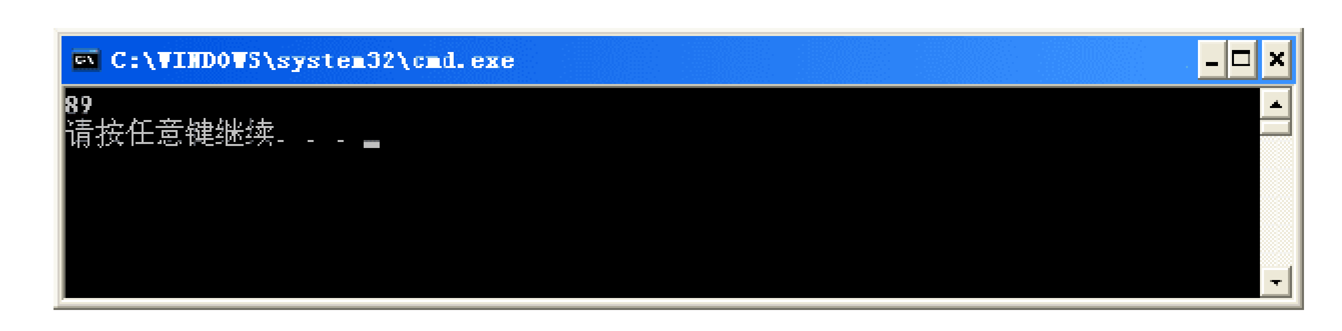


图 4-1　获取一个成绩

5. 实例分析



源码解析：

在该案例中，初始化 score 数组，中括号[]里面的数字 5 表示这个数组包含 5 个元素，分

别是大括号{}里面的 95, 65, 89, 75, 63 这 5 个元素的值。获取索引为 2 的元素值，并用 Console.WriteLine()方法将元素值显示在控制台上。需要注意的一点就是，索引必须小于元素的个数。

4.1.2 获取最高成绩

获取数组中最大的元素值。

 **视频教学**：光盘/videos/4/Array.avi　　　　长度：6 分钟

1. 基础知识——数组的排序

在使用数组之前，首先要初始化数组。

(1) 初始化数组——直接赋值。

将一组值(被包括在大括号之内，每一个值使用逗号分隔)直接赋给数组。将一组值{95, 65, 89, 75, 63}赋给 score 数组。该数组初始化之后，它的长度为 5，元素的值分别为 95、65、89、75 和 63。代码如下：

```
int []score = {95, 65, 89, 75, 63};
```

(2) 冒泡排序。

要想获取数组中最大的那个值，就要先对数组进行排序。冒泡排序算法有两种：一是单向冒泡排序算法；二是双向冒泡排序算法。下面就讲解一下单向冒泡排序算法中的一种，也是比较常用的冒泡排序算法——从上向下冒泡的冒泡排序。基本思路如下。

① 首先将第一个记录的关键字和第二个记录的关键字进行比较，若为“逆序”(即 L.r[1].key>L.r[2].key)，则将两个记录交换之，然后比较第二个记录和第三个记录的关键字。依次类推，直至第 n−1 个记录的关键字和第 n 个记录的关键字比较过为止。这是第一趟冒泡排序，其结果是使得关键字最大的记录被安置到最后一个记录的位置上。

② 然后进行第二趟冒泡排序，对前面的 n−1 个记录进行同样的操作，其结果是使关键字次大的记录被安置到第 n−1 个记录的位置。

一般地，第 i 趟冒泡排序是从 L.r[1]到 L.r[n−i+1]依次比较相邻两个记录的关键字，并在“逆序”时交换相邻记录，其结果是这 n−i+1 个记录中关键字最大的记录被交换到第 n−i+1 的位置上。整个排序过程需要进行 k(1≤k＜n)趟冒泡排序，显然，判别冒泡排序结束的条件应该是“在一趟排序过程中没有进行过交换记录的操作”。

算法描述如下：

```
int []score = new int[5] { 95, 65, 89, 75, 63 };
for (int i=1; i<score.Length; i++)
{
    for (int j=0; j<i; j++)
    {
```

```
        if (score[j] > score[j+1])
        {
            int tmp = score[j];
            score[j] = score[j+1];
            score[j+1] = tmp;
        }
    }
}
```

2. 实例描述

有一只乌鸦不好看，但特别聪明，智慧过人。一天，它干完活又累又渴，真想喝水。忽然，它看一只大水罐，满心欢喜。它飞到水罐旁，一看罐里的水不多了，嘴探进去也喝不到，怎么办呢？它使劲地用身体撞水罐，又用翅膀推水罐，想把水罐弄倒，好喝水。可是水罐又大又重，它的力量太小了，弄不倒这罐子。忽然，它急中生智，可以叼些石子放到罐里，石子多了，罐子里的水不就升高了吗？这么想，就这么做了，它不厌其烦地一块一块地用嘴叼石子，功夫不负用功的乌鸦，终于放了很多石子，水上升了，它就喝到了水。它痛痛快快地喝了个够，解了渴。嗯，这是小时候听过的故事，在这里回忆一下。

冒泡排序也是同样的道理，下面这个案例中将为大家讲解从下到上的冒泡排序算法。

3. 实例应用

【例 4-2】获取最高成绩。

(1) 首先初始化一个数组，名为 score。示例代码如下：

```
int []score = { 95, 65, 89, 75, 63 };
```

(2) 将数组 score 冒泡排序，这里用到的排序方法是从下到上冒泡的排序。示例代码如下：

```
for (int i=1; i<score.Length; i++)
{
    for (int j=0; j<i; j++)
    {
        if (score[j] > score[j+1])
        {
            int tmp = score[j];
            score[j] = score[j+1];
            score[j+1] = tmp;
        }
    }
}
```

(3) 排序后的数组值是倒序排列的，所以只需将数组的最后一个值输出即可。代码如下：

```
Console.WriteLine(score[score.length-1]);
```

4. 运行结果

运行程序，效果如图 4-2 所示。



图 4-2　获取最高成绩

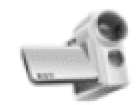
5. 实例分析


源码解析：

在该案例中，初始化 score 数组，包含 95, 65, 89, 75, 63 这 5 个元素的值。然后将数组进行了从下到上的冒泡排序算法，这个排序为顺序，所以，数组的最后一个值就是所要获取的最高成绩。

4.1.3　降序输出成绩

将初始化的成绩数组降序排列，并一一显示在控制台上。


视频教学：光盘/videos/4/Array_method.avi　　　　长度：5 分钟

1. 基础知识——数组的遍历与排序

(1) 初始化成绩数组——使用 new 操作符，省略数组的长度。

将一组值{95, 65, 89, 75, 63}赋给 score 数组。该初始化操作使用了 new 操作符，但是省略了数组的长度。该数组初始化之后，它的长度为 5，它的元素的值分别为 95、65、89、75 和 63：

```
int []score = new int[] {95, 65, 89, 75, 63};
```

(2) 遍历数组。

遍历数组是指依次访问数组中的每个元素的操作。访问数组中的元素最常用的方法是使用下标。根据是否使用下标，遍历数组的方式分为两种。第一种，使用下标遍历，如 for、while、do ... while 等语句。第二种，不使用下标遍历，如 foreach 语句。

(3) 冒泡排序——从下向上冒泡的冒泡排序。

① 首先将第 n−1 个记录的关键字和第 n 个记录的关键字进行比较，若为“逆序”(即 L.r[n].key<L.r[n−1].key)，则将两个记录交换之，然后比较第 n−2 个记录和第 n−1 个记录的关键字。依次类推，直至第 1 个记录的关键字和第 2 个记录的关键字比较过为止。这是第一趟起泡排序，其结果是使得关键字最小的记录被安置到第一个记录的位置上。

② 然后进行第二趟起泡排序，对后面的 n−1 个记录进行同样的操作，其结果是使关键字次小的记录被安置到第 2 个记录的位置。

一般地，第 i 趟起泡排序是从 L.r[n] 到 L.r[i]依次比较相邻两个记录的关键字，并在“逆

序”时交换相邻记录，其结果是这 n-i+1 个记录中关键字最小的记录被交换到第 i 的位置上。整个排序过程需要进行 k(1≤k<n)趟起泡排序，显然，判别起泡排序结束的条件应该是“在一趟排序过程中没有进行过交换记录的操作”。

算法描述如下：

```
int []score = new int[5] { 95, 65, 89, 75, 63 };
for (int i=1; i<score.Length; i++)
{
    for (int j=0; j<i; j++)
    {
        if (score[j] < score[j+1])
        {
            int tmp = score[j];
            score[j] = score[j+1];
            score[j+1] = tmp;
        }
    }
}
```

2. 实例描述

前面已经讲到了一种排序的算法。下面这个案例，我们来学习另外一种排序算法。说到这个排序，就让人想起了排队，坐公交车要排队，买饭要排队，看病要排队，确实，凡事都得有个先来后到的。来看下面的这个排序案例吧。

3. 实例应用

【例 4-3】降序输出成绩。

(1)　首先初始化一个成绩数组 score。示例代码如下：

```
int []score = new int[] {95,65,89,75,63};
```

(2)　对 score 数组进行降序排列。示例代码如下：

```
int []score = new int[5] { 95, 65, 89, 75, 63 };
for (int i=1; i<score.Length; i++)
{
    for (int j=0; j<i; j++)
    {
        if (score[j] < score[j+1])
        {
            int tmp = score[j];
            score[j] = score[j+1];
            score[j+1] = tmp;
        }
    }
}
```

(3)　最后将数组用 for 语句遍历输出，显示在控制台上。示例代码如下：

```
for (int i=0; i<score.Length; i++)
{
    Console.WriteLine(score[i]);
}
```

4. 运行结果

运行此程序，效果如图 4-3 所示。



图 4-3　降序输出成绩

5. 实例分析



源码解析：

在该案例中初始化 score 数组，包含 95,65,89,75,63 这 5 个元素的值。然后将数组进行从下到上的冒泡排序算法，这个排序为倒序，最后将数组用 for 语句遍历输出。

4.1.4　二维成绩数组

除了一维数组，二维数组也是数组中最常用的，它的显示形状为矩形，也可以理解为表格。



视频教学：光盘/videos/4/DuoWei.avi　　长度：9 分钟

1. 基础知识——二维数组

二维数组是指维度数为 2 的数值。声明二维数组的语法如下：

```
type []arrName;
```

其中，type 表示数组元素的类型，如 int、double、string、object 等，中括号里存在一个逗号，表示该数组为二维数组。

二维数组又称为矩阵，行列数相等的矩阵称变方阵。对称矩阵 a[i][j] = a[j][i]，对角矩阵：n 阶方阵的所有非零元素都集中在主对角线上。

2. 实例描述

二维数组就像是表格一样呈现在眼前，每一个元素的值就相当于表格中单元格的值。

3. 实例应用

【例 4-4】二维成绩数组。

(1)　首先，初始化一个二维数组 score。示例代码如下：

```
int [,]score = new int[,] { { 45, 78, 56, 24, 32, 39 }, { 45, 78, 56, 24, 32,
39 }, { 45, 78, 56, 24, 32, 39 } };
```

(2) 将 score 数组的每个元素的值显示在控制台上。示例代码如下：

```
for (int i=0; i<3; i++)
{
    for (int j=0; j<6; j++)
    {
        Console.Write(score[i, j] + "  ");
    }
    Console.WriteLine();          //换行
}
```

4. 运行结果

运行此程序，效果如图 4-4 所示。



图 4-4　二维成绩数组

5. 实例分析



源码解析：

初始化二维数组 score，显示为 3 行 5 列。用 for 语句嵌套 for 语句输出每行元素的值。

4.2　使用集合列表

随着.NET Framework 3.5 的逐渐扩展，在 Web 项目的实际开发中，经常会用到集合列表，用于存放数据。本节中，就会讲到集合列表的声明及使用方法。

 视频教学：光盘/videos/4/Array_method.avi　　　　长度：5 分钟

4.2.1　添加一本图书到集合

添加一本图书到集合，其实就是将图书的名字添加到集合列表。

1. 基础知识——List的基本操作

集合列表的命名空间是 System.Collections.Generic。

(1) 定义集合列表

语法如下：

```
List<T> ListName;
```

其中 T 代表将要存放的数据的类型。实例化这个集合列表，代码如下：

```
List<T> ListName = new List<T>();
```

(2) 添加集合元素

使用 Add()方法可以给列表添加元素。括号里面填写元素的值，类型要与<>括号里的类型一致。代码如下：

```
List<int> lst = new List<int>();
lst.Add(1);
lst.Add(2);
```

(3) 输出集合元素

用 foreach 语句获取集合中的元素。代码如下：

```
foreach (int i in lst)
{
    Console.WriteLine(i);
}
```

2. 实例描述

仓库里存放着很多种商品，每一种商品都有它自己的存放空间，便于仓库人员的管理。那么在下面的这个案例中，将会讲到一个类似于仓库的类——泛型类 List<T>，它是集合列表。

3. 实例应用

【例 4-5】添加一本图书到集合中。代码如下：

```
List<string> book = new List<string>();
book.Add("格林童话");
foreach (string i in book)
{
    Console.WriteLine(i);
}
```

4. 运行结果

运行此程序，效果如图 4-5 所示。

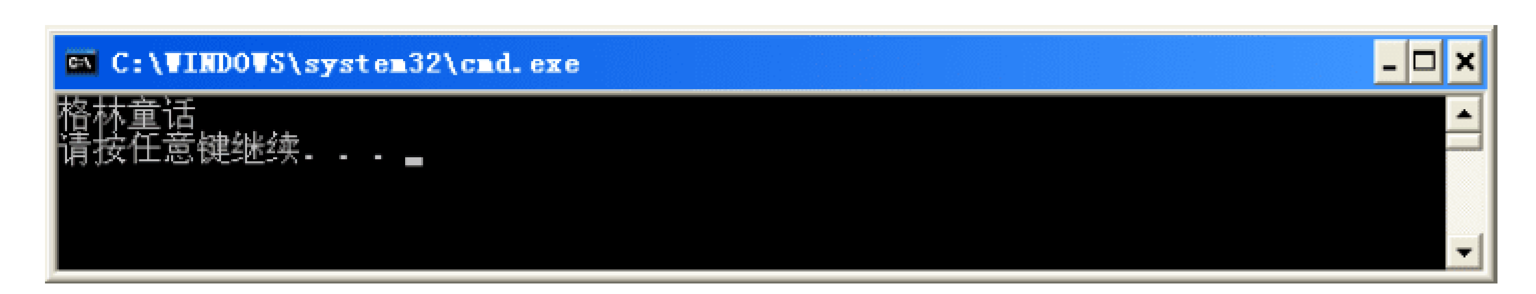


图 4-5　添加一本图书到集合中

5. 实例分析



源码解析：

List 集合里面保存的数据是 string 类型，用于保存图书的名称。这里实例化一个泛型集合，类型为 string。泛型集合中的每一个元素用 foreach 语句将 book 泛型集合里的图书显示在控制台上。

4.2.2　按价格升序输出排列集合

每一本图书的价格不一样，为了方便管理，可以将图书按价格排序。

1. 基础知识——集合列表的排序

List<T>类可以对元素排序。Sort()方法定义了几个重载方法。可以传送给它的参数有泛型委托 Comparison<T>、泛型接口 Icomparer<T>、泛型接口 Icomparer<T>和一个范围值。一般最常用的就是 Sort()不带参数的方法。

2. 实例描述

从我国古代开始，就在为各种各样的事物排序了，最为明显的就是古代的科举考试，因此就有了“状元”、“榜眼”、“探花”的代名词。这个小节中，就一起来鉴定一下集合列表的排序方法。

3. 实例应用

【例 4-6】按价格升序输出排列集合。

(1) 实例化一个保存了价格的集合列表，命名为 price，数据类型为 double。示例代码如下：

```
List<double> price = new List<double>();
```

(2) 向 book 集合列表中添加数据。示例代码如下：

```
price.Add(12.50);
price.Add(25.00);
price.Add(33.20);
```

(3) 用 Sort()方法将列表排序。示例代码如下：

```
price.Sort();
```



默认的 Sort()方法是按升序排列。

(4) 用 foreach 语句将集合列表遍历输出。示例代码如下：

```
foreach (double pri in price)
{
    Console.WriteLine(pri);
}
```

4. 运行结果

运行此程序，效果如图 4-6 所示。

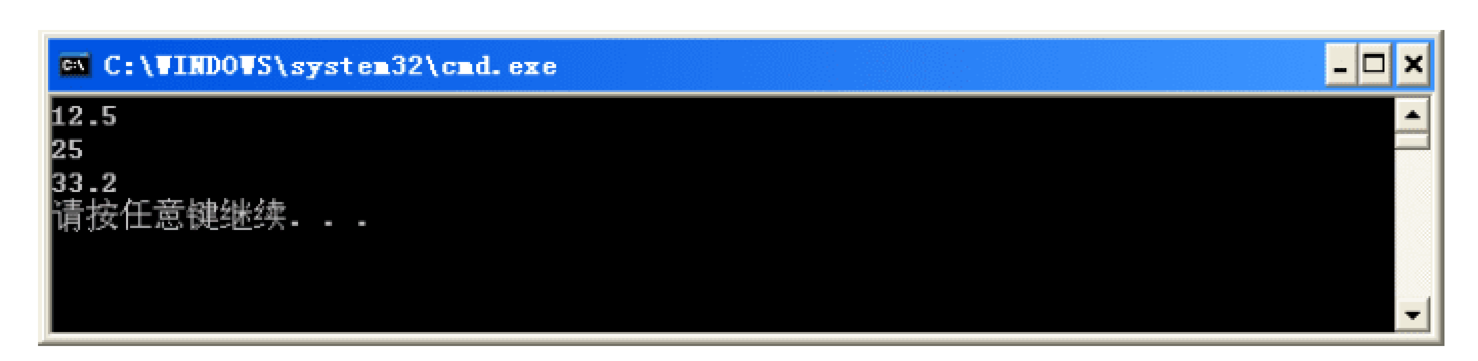


图 4-6 按价格升序输出排列集合

5. 实例分析



源码解析：

这里向实例化之后的集合列表 price 添加了 3 条数据，数据类型为 int。用到的一个关键点就是 Sort()排序方法，它默认为升序。最后将排序后的列表用 foreach 语句遍历输出。

4.2.3 在集合中查找图书

与访问数组元素类似，都是获取集合中的某个值。

1. 基础知识——集合的索引

访问集合中的元素，是通过下标来访问的。语法如下：

```
arrName[int]
```

其中 arrName 表示实例化之后的集合的名字，int 表示索引值，比如 arrName[2]，表示 arrName 集合索引为 2 的索引值，即集合中第 3 个元素的值。

2. 实例描述

在前面的章节中，我们已经讲过如何访问数组元素。那么在本节中，我们将要讲到如何访问集合元素。

3. 实例应用

【例 4-7】在集合中查找图书。

```
List<string> book = new List<string>();
book.Add("格林童话");
book.Add("公主日记");
book.Add("奥特曼");
Console.WriteLine(book[2]);
```

4. 运行结果

运行此程序，效果如图 4-7 所示。

```
C:\WINDOWS\system32\cmd.exe
奥特曼
请按任意键继续. . .
```

图 4-7　在集合中查找图书

5. 实例分析


源码解析：

首先，实例化集合列表 book。然后，向 book 集合列表中添加 3 条数据。访问 book 集合列表中第 3 个元素，也就是索引为 2 的元素值，并将它显示在控制台上。

4.3　使用日期和时间

使用日期和时间是 C#编程中最为常见的操作之一。在.NET Framework 中，与日期和时间相关的结构为 DateTime 结构和 TimeSpan 结构。DateTime 结构表示时间上的某一刻，TimeSpan 表示时间间隔。本节将讲解 C#编程中如何使用日期和时间，主要介绍以下知识点：

- 获取当前完整日期和时间
- 格式化时间
- 计算时间的差
- 追加时间

通过本节的学习，读者将可以了解 DateTime 和 TimeSpan 结构，以及使用这两个结构实现操作日期和时间的常用方法。


视频教学：光盘/videos/4/Date.avi　　　　长度：5 分钟

4.3.1　获取当前完整日期和时间

获取日期和时间，在 Web 项目开发过程当中，是经常要用到的。比如说，当用户登录系统之后，需要记录用户的登录时间。对于项目来说，增加了安全性能。

1. 基础知识——DateTime结构

获取当前完整的日期和时间，需要了解 DateTime 结构。

DateTime 是一个结构，且为值类型，表示时间点，包括日期和时间两部分。DateTime 表示的日期和时间由年、月、日、时、分、秒，以及分隔符组成。最常用的一种格式如下所示：

```
yyyy-mm-dd hh:mm:ss
```

其中，yyyy 表示年，为 4 位数字；MM 表示月，为 2 位数字；dd 表示日，为 2 位数字；

hh 表示时，为 2 位数字；mm 表示分，为 2 位数字；ss 表示秒，为 2 位数字；“-”、“ ”和“:”都为分隔符。

DateTime 结构包含 14 个常用属性。其中，Now、Today 和 UtcNow 为静态属性，其他属性为实例属性。通过这 14 个属性，可以获取时间的值或时间某一个部分的值，如年份、月份、日期、小时、分钟、秒等。DateTime 结构的属性具体说明如表 4-1 所示。

表 4-1　DateTime结构的常用属性表

属　性	说　明
Now	静态属性，获取此计算机上的当前时间
Today	静态属性，获取当前日期
UtcNow	静态属性，获取此计算机上的当前时间，并表示为协调通用时间(UTC)
Year	获取年份部分
Month	获取月份部分
Day	获取为该月中的第几天
Hour	获取小时部分
Minute	获取分钟部分
Second	获取秒部分
Ticks	获取日期和时间的刻度数
Date	获取日期部分
DayOfWeek	获取星期几
DayOfYear	获取该年中的第几天
TimeOfDay	获取当天的时间，即当天自午夜以来已经过时间的部分



DateTime 结构包括的 16 个属性均为只读属性。

2. 实例描述

获取时间这个东西是很强大、很广泛的，银行打印的凭条显示有当前时间，超市的发票也有当前时间，甚至是有些网站也显示当前时间，这很特殊，就像闹钟一样，一直在走。

这里就来一起获取当前时间吧。

3. 实例应用

【例 4-8】获取当前完整日期和时间。代码如下：

```
Console.WriteLine(DateTime.Now.ToString());
```

4. 运行结果

运行此程序，效果如图 4-8 所示。



图 4-8　获取当前完整日期和时间

5. 实例分析



源码解析：

使用 DateTime 结构的 Now 属性，可获取当前完整的日期和时间，并显示在控制台上。

4.3.2　格式化时间

格式化时间就是将时间以不同格式的字符串形式输出。

1. 基础知识——格式化

日期和时间格式化是指将DateTime结构的实例转换为其相对应的某一种字符串表示形式。DateTime 结构提供了多个以 To 开头的方法，这些方法可以将 DateTime 对象转换为不同格式的字符串形式，常用的方法如表 4-2 所示。

表 4-2　DateTime结构中有关格式化的方法表

方　法	说　明
ToString()	转换为其等效的字符串表示形式
ToLongDateString()	转换为其等效的长日期字符串表示形式
ToShortDateString()	转换为其等效的短日期字符串表示形式
ToLongTimeString()	转换为其等效的长时间字符串表示形式
ToShortTimeString()	转换为其等效的短时间字符串表示形式

2. 实例描述

以前做项目的时候，都是获取当前完整的日期和时间，那么，在这个案例中，将为大家展示一下如何以不同格式的字符串形式输出时间。

3. 实例应用

【例 4-9】格式化时间。

(1)　获取当前日期的长日期的字符串形式。示例代码如下：

```
Console.WriteLine("长日期：");
Console.WriteLine(DateTime.Now.ToLongDateString());
```

(2)　获取当前日期的短日期的字符串形式。示例代码如下：

```
Console.WriteLine("短日期：");
Console.WriteLine(DateTime.Now.ToShortDateString());
```

(3) 获取当前日期的长时间的字符串形式。示例代码如下：

```
Console.WriteLine("长时间：");
Console.WriteLine(DateTime.Now.ToLongTimeString());
```

(4) 获取当前日期的短时间的字符串形式。示例代码如下：

```
Console.WriteLine("短时间：");
Console.WriteLine(DateTime.Now.ToShortTimeString());
```

4. 运行结果

运行此程序，效果如图 4-9 所示。



图 4-9 格式化时间

5. 实例分析



源码解析：

获取当前时间，有很多种表现形式。ToString()方法表示获取当前完整的日期和时间，在做项目的过程当中，这个是最为常用的。还有其他的一些方法，比如 ToLongDateString()、ToShortDateString()、ToLongTimeString()、ToShortTimeString()等。

4.3.3 考试倒计时——计算时间的差

考试倒计时，在某个时间点开始考试，在经过多长时间之后的某个时间点结束考试。

1. 基础知识——TimeSpan结构

TimeSpan 结构表示一个时间间隔或持续时间，可以按照正负天数、小时数、分钟数、秒数以及秒的小数部分进行度量。

TimeSpan 值表示为[-]d.hh:mm:ss:ff 格式的字符串。减号(可选)表示负时间间隔；d 表示天数。TimeSpan 结构包含 6 个常用实例属性。通过这 6 个常用属性，可以获取 TimeSpan 对象的天数、小时数、分钟数、秒数、毫秒数等。TimeSpan 结构的属性具体说明如表 4-3 所示。

表 4-3　TimeSpan结构的常用属性表

属　性	说　明
Days	获取 TimeSpan 对象的整天数
Hours	获取 TimeSpan 对象的整小时数
Minutes	获取 TimeSpan 对象的整分钟数
Ticks	获取 TimeSpan 对象的刻度数
Seconds	获取 TimeSpan 对象的整秒数
Milliseconds	获取 TimeSpan 对象的整毫秒数

TimeSpan 结构包含 6 个常用的实例方法，如表 4-4 所示。

表 4-4　TimeSpan结构的实例方法表

方　法	说　明
Add()	将当前实例的值加上指定的 TimeSpan 的值
Subtract()	从当前实例中减去指定的 TimeSpan 的值
Negate()	获取当前实例的值的符号相反、值相等的 TimeSpan
Duration()	获取当前实例的值的绝对值
CompareTo()	将当前实例与指定的对象或 TimeSpan 进行比较
ToString()	返回当前实例的值的字符串表示形式

2. 实例描述

以前做过一个叫在线考试系统的项目，里面就有考试倒计时的小功能，当考生登录到考试题这个界面，并点击“开始考试”时，就开始倒计时了。其实，像比赛也是一样，从开始比赛的那一刻起，倒计时开始。

这里就简单为大家讲一下考试倒计时。

3. 实例应用

【例 4-10】考试倒计时——计算时间的差。

计算时间差可以使用 Subtract()方法。该方法可以计算两个时间的差值，也可以计算时间和时间间隔的差值。它存在以下两种重载形式：

- public TimeSpan Subtract(DateTime value)：从当前实例中减去指定的日期和时间，返回类型为 TimeSpan。其中，value 参数表示被减去的日期和时间。
- public DateTime Subtract(TimeSpan value)：从当前实例中减去指定的时间间隔，返回类型为 DateTime。其中，value 参数表示被减去的时间间隔。

(1) 比如某学校期末考试时间是 12 月 1 日，定义时间变量，并为其赋值。示例代码如下：

```
DateTime tim = DateTime.Parse("2010-12-1");
```

(2) 获取当前日期，示例代码如下：

```
DateTime tim2 = DateTime.Now.Date;
```

(3) 计算两个日期的差，即为考试倒计时。示例代码如下：

```
TimeSpan span = tim2.Subtract(tim).Duration();
Console.WriteLine("考试倒计时" + span.Days + "天");
```

4. 运行结果

运行此程序，效果如图 4-10 所示。



图 4-10　考试倒计时——计算时间的差

5. 实例分析



源码解析：

计算 tim2 距离 tim 还有多长时间，这里以“天”计算。Duration()获取两个时间差的绝对值，即为距离考试的天数。Subtract()多用于重大节日或事件的提示。

4.3.4　追加时间

追加时间，其实就是将时间延长。

1. 基础知识——追加

追加时间是指将指定的时间或时间部分(如年份、月份、小时等)追加到一个时间上，并计算出一个新的时间。DateTime 结构提供多个以 Add 开头的方法可以实现追加时间的功能，这些方法的具体说明如表 4-5 所示。

表 4-5　DateTime结构中与追加时间相关的方法表

方　法	说　明
Add()	将指定的 TimeSpan 的值加到指定的时间上
AddYears()	将指定的年份数加到指定的时间上
AddMonths()	将指定的月份数加到指定的时间上
AddDays()	将指定的天数加到指定的时间上
AddHours()	将指定的小时数添加到指定的时间上
AddMinutes()	将指定的分钟数添加到指定的时间上
AddSeconds()	将指定的秒数添加到指定的时间上
AddMilliseconds()	将指定的毫秒数添加到指定的时间上
AddTicks()	将指定的刻度数添加到指定的时间上

2. 实例描述

每逢过年过节，火车站更是人多拥挤，进了车站之后，就是漫长的等待，等到火车快来的时候，人们就开始排队进站了，好不容易该进站了，铁路人员来了一句“T168 次列车晚点 30 分钟，将于 23 点 30 分到站”。下面的例子，就为大家讲解一下如何追加时间。

3. 实例应用

【例 4-11】追加时间。代码如下：

```
DateTime date = DateTime.Now.AddDays(2);
Console.WriteLine(date);
```

4. 运行结果

运行此程序，效果如图 4-11 所示。

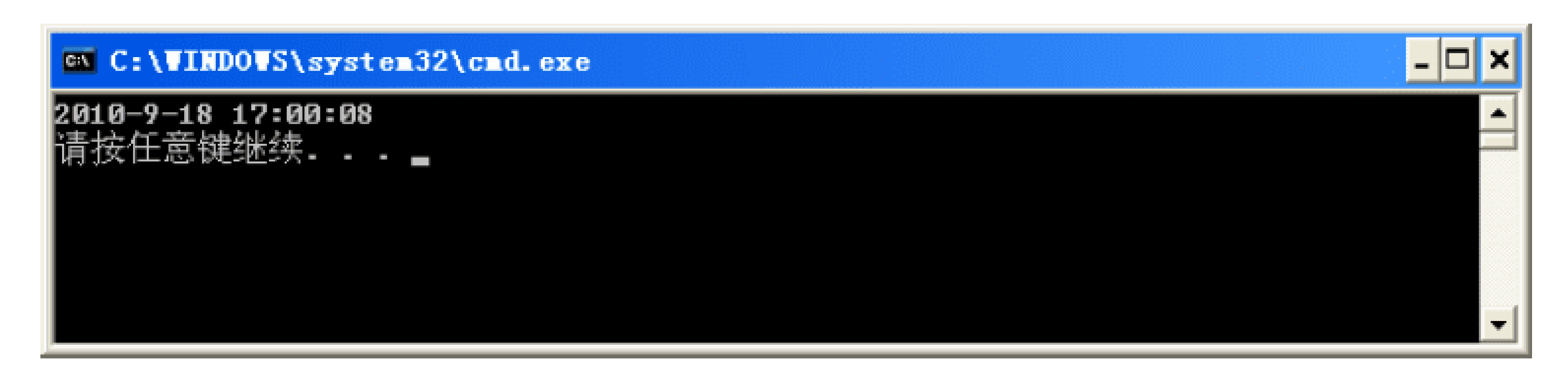


图 4-11　追加时间

5. 实例分析



源码解析：

调用 AddDays()方法将 2 天加到当前时间上，并计算出一个新的时间，DateTime 结构中包含有关追加时间的一些常用方法，如 AddDays()、Add()、AddMonths()等。

4.4　使用字符串

字符串类型是 C#编程语言中最为常用的一种数据类型，.NET Framework 为字符串提供了丰富的操作和处理方法。通过这些方法，开发人员可以简单轻松地处理字符串。本节将讲解.NET Framework 中处理字符串的方法，主要介绍以下两个知识点：

- 不变字符串和可变字符串
- 字符串的常用操作

通过本节的学习，读者将可以了解.NET Framework 中处理字符串的方法，并为以后开发功能强大的、高效的 C#应用程序奠定坚实的基础。

4.4.1 不变字符串和可变字符串

不变与可变就只有一字之差，从表面来看，也很容易理解。通俗地说，一个不可以加长，另一个可以加长。但是，读者通过这里的学习，就会有更深一步的了解。

 视频教学：光盘/videos/4/string_bijiao.avi　　长度：6 分钟

1. 基础知识——String类和StringBuilder类

在.NET Framework 中，字符串可以分为两种类型：不变字符串(由 String 类表示)和可变字符串(由 StringBuilder 类表示)。不变字符串一旦被创建，那么它的内容是不能被修改的。而可变字符串被创建之后，它的内容是可以动态修改的。

(1) String 类

String 类表示一系列的有序 Unicode 字符，用来表示文本。String 对象是字符(Char 对象)的有序集合，用来表示字符串。String 对象的值是该有序集合的内容，并且该值是不可变的，也就是说一旦创建了一个 String 对象，就不能修改该对象的值。

虽然有些方法看起来修改了 String 对象，而实际上是返回了一个包含修改后内容的新的 String 对象，系统会重新为它分配一块内存。如果频繁地修改 String 对象，则可能产生大量的不可访问的内存，会影响应用程序的性能。比如说，声明一个名为 Str 的 String 对象，关于内存分配的问题，效果如图 4-12 所示。

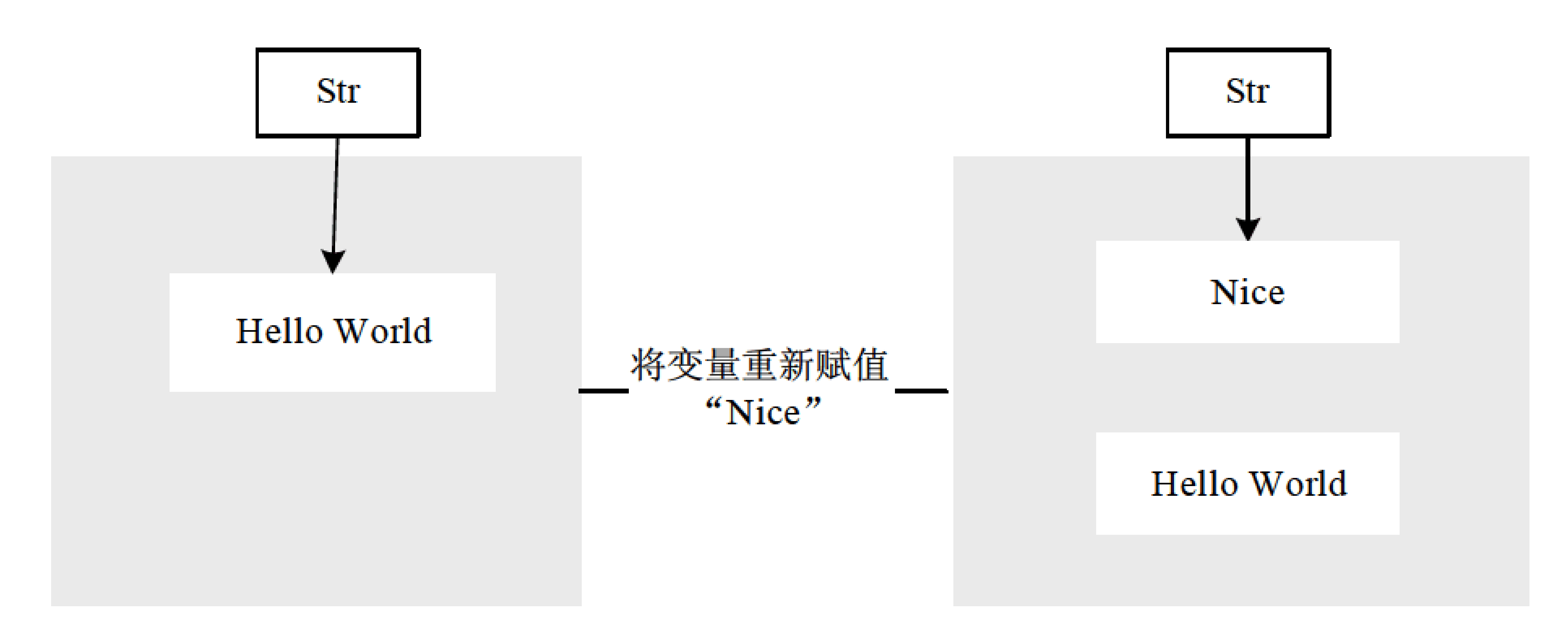


图 4-12　Str变量及为Str变量赋值之后的内存分配

(2) StringBuilder 类

StringBuilder 类表示可变字符串，可以创建能够动态修改(如追加、移除、替换、插入等)的字符串对象。由于 StringBuilder 对象的值占用的内存是可修改的，即对 StringBuilder 对象每一次的修改操作都可能不需要分配新的内存，而在该对象的值所占的内存上进行操作，从而减少了应用程序的开销。

StringBuilder 类包括 4 个常用属性，具体说明如表 4-6 所示。

表 4-6　StringBuilder类的常用属性

属　性	说　明
Length	获取或设置 StringBuilder 对象的长度
Capacity	获取或设置 StringBuilder 对象的容量
MaxCapacity	获取或设置 StringBuilder 对象的最大容量
Chars	获取或设置指定位置处的字符



通常情况下，如果 StringBuilder 实例没有指定容量或最大容量，那么该实例具有一个默认容量值。当然，开发人员也可以通过 Capacity 属性设置实例的容量，被设置的容量值必须大于或等于其 Length 属性的值。

2. 实例描述

玩游戏能让人进入紧张状态，因为它富有挑战性。在作者看来，游戏可以分为两大类。一类是重新开始；另一类是接着闯关。当然，前提是死翘翘了。如果重新开始，就要从第一关开始，这种感觉会消磨掉人的耐性的；如果可以在当前战场重新闯关的话，就可以更上一层楼了，让人有种成就感。那么，不变字符串和可变字符串的道理也是一样的。

下面的例子，就会为大家展示一下不变字符串和可变字符串的原理。

3. 实例应用

【例 4-12】不变字符串和可变字符串。

(1) 声明一个名称为 str 的 String 对象，它的值为“Hello World”。然后，对 str 变量进行赋值，这个值为“Good By”。示例代码如下：

```
string str = "Hello World";
str = "Good By";
Console.WriteLine(str);
```

(2) 实例化一个名称为 StrSb 的 StringBuilder 对象，它的值默认为空字符串。示例代码如下：

```
StringBuilder StrSb = new StringBuilder();
StrSb.Append("Nice To Meet You");
StrSb.Append("  Too");
Console.WriteLine("输出StringBuilder对象的值: " + StrSb);
```

4. 运行结果

运行此程序，效果如图 4-13 所示。



图 4-13　不变字符串和可变字符串

5. 实例分析



源码解析：

该案例中，str = "Good By";为 str 重新赋值，其实就是创建一个新的 String 对象，系统为 str 变量重新分配了一块内存，并存放 "Good By"。这里的 StringBuilder 对象——StrSb 默认为空字符串，然后用 Append()方法追加字符串 "Nice To Meet You"，再一次追加 " Too"，最后 StrSb 的值为 "Nice To Meet You Too"。这里就说明 StringBuilder 对象是可变长字符串。

4.4.2 字符串的处理

在许许多多、大大小小的项目开发过程当中，字符串的处理起到了桥梁的作用。



视频教学：光盘/videos/4/string_op.avi　　长度：6 分钟

1. 基础知识——字符串的常用操作

字符串的一些常用操作包括有插入字符串、替换字符串、移除字符串、分隔字符串、填充字符串、连接字符串、获取子字符串、查找字符串等。下面我们就以 String 类为例，为大家讲解字符串的一些常用操作。

(1) 插入字符串

插入字符串是将新的字符串插入到字符串的指定位置，该功能由 Insert()方法实现。Insert()的使用方法如下：

```
public string Insert(int startIndex, string value);
```

上述方法将 value 参数指定的字符串插入到字符串的 startIndex 指定的位置。

startIndex 参数表示插入的位置(即索引值)，value 参数表示被插入的新字符串。上述方法将返回一个新的字符串。

(2) 替换字符串

替换字符串是指将字符串中的指定字符替换为新的字符，或者将指定的字符串替换为新的字符串，该功能由 Replace()方法实现。Replace()方法的两个重载方式如下。

- public string Replace(char oldChar, char newChar)：将字符串中指定的字符替换为新的字符。
- public string Replace(string oldValue, string newValue)：将字符串中指定的字符串替换为新的字符串。

(3) 移除字符串

移除字符串是将指定位置的字符或者字符串从字符串中移除。该功能由 Remove()方法实现。Remove()方法的两个重载方式如下。

- public string Remove(int startIndex)：从 startIndex 位置开始移除其后的所有字符。
- public string Remove(int startIndex, int count)：从 startIndex 位置开始移除其后的数量为

Web 程序员成功之路

count 的字符。

(4) 分隔字符串

分隔字符串是将一个字符串用指定的分隔字符或字符串分隔为多个子字符串，并返回由子字符串组成的字符串数组。该功能由 Split()方法实现，它的几个重载方式如下：

```
public string[] Split(params char []separator);
public string[] Split(char []separator, int count);
public string[] Split(char []separator, StringSplitOptions options);
public string[] Split(string []separator, StringSplitOptions options);
public string[] Split(char []separator, int count, StringSplitOptions options);
public string[] Split(string []separator, int count, StringSplitOptions
options);
```

separator 参数表示分隔字符或字符串数组。count 参数指定返回的子字符串的最大数量。options 参数指定字符串分隔选项，它的类型为 StringSplitOptions 枚举，包含以下两个值。

- None：子字符串的数组可以包含空字符串。
- RemoveEmptyEntries：子字符串数组不包含空字符串。

(5) 填充字符串

填充字符串是向字符串的指定位置填充指定的字符，该功能由 String 类的 PadRight()方法和 PadLeft()方法实现。其中，PadRight()方法表示在字符串的右边填充字符，PadLeft()方法表示在字符串的左边填充字符。由于这两个方法比较相似，下面将只介绍 PadLeft()方法，它的两个重载方式如下：

- public string PadRight(int totalWidth)：在字符串的左边填充空白字符。新字符串的长度为 totalWidth。
- public string PadLeft(int totalWidth, char paddingChar)：左对齐此字符串中的字符，在右边用空格填充以达到指定的总长度。

totalWidth 参数表示填充之后字符串的总长度，paddingChar 表示被填充的字符。

(6) 连接字符串

连接字符串是将多个字符串连接为一个新的字符串。用 Concat()或 Join()方法实现，这里以 Concat()方法为例，该方法的几个重载方式如下：

```
public static string Concat(object arg0);
public static string Concat(params object []args);
public static string Concat(object arg0, object arg1);
public static string Concat(string str0, string str1);
public static string Concat(object arg0, object arg1, object arg2);
```

arg0 参数表示被连接的对象，其他参数表示被连接的字符串。

(7) 获取子字符串

获取子字符串是指从字符串中获取指定的子字符串。用 SubString()方法实现，该方法的两个重载方式如下：

```
public string Substring(int startIndex);
public string Substring(int startIndex, int length);
```

startIndex 参数表示指定子字符串开始的位置，length 参数表示指定子字符串的最大长度。

(8) 查找字符串

查找字符串是从字符串中查找指定的字符或子字符串。用 IndexOf()方法实现，返回值为 int 类型，表示索引位置。该方法的几个常用的重载方式如下：

```
public int IndexOf(char value);
public int IndexOf(string value);
public int IndexOf(char value, int startIndex);
public int IndexOf(string value, int startIndex);
```

value 参数表示将要查找的字符或字符串，startIndex 参数表示要查找字符或字符串的指定位置。

2. 实例描述

民间有一种艺术叫做变脸，一遮脸就是不同的模样，表演得神乎其神。当然了，其中肯定是有窍门儿的，至于窍门儿的所在，就不得而知了。

可是呢，我们也知道怎样在程序中实现“变脸”。这里就为大家讲解一下有关字符串的处理——字符串的一些常用操作。

3. 实例应用

【例 4-13】字符串的处理。

(1) 插入字符串。示例代码如下：

```
Console.WriteLine("插入字符串：");
string Article = "abcd";
string Article2 = Article.Insert(2, "33");
Console.WriteLine("原字符串：" + Article);
Console.WriteLine("新字符串：" + Article2);
```

(2) 替换字符串。示例代码如下：

```
Console.WriteLine("替换字符串：");
string tihuan = "abcd";
string tihuan2 = tihuan.Replace('b', 'd');
Console.WriteLine("原字符串：" + tihuan);
Console.WriteLine("新字符串：" + tihuan2);
```

(3) 移除字符串。示例代码如下：

```
Console.WriteLine("移除字符串：");
string yichu = "abcd";
string yichu2 = yichu.Remove(0, 1);
Console.WriteLine("原字符串：" + yichu);
Console.WriteLine("新字符串：" + yichu2);
```

(4) 分隔字符串。示例代码如下：

```
Console.WriteLine("分隔字符串: ");
string fenge = "a,b,c,d";
string []fenge2 = fenge.Split(new char[]{','});

foreach (string s in fenge2)
{
    Console.WriteLine(s);
}

Console.WriteLine("原字符串: " + fenge);
Console.WriteLine("分隔后的字符串: " + fenge2);
```

(5) 填充字符串。示例代码如下：

```
Console.WriteLine("填充字符串: ");
string tianchong = "abcd";
string tianchong2 = tianchong.PadLeft(4);
Console.WriteLine("原字符串: " + tianchong);
Console.WriteLine("新字符串: " + tianchong2);
```

(6) 连接字符串。示例代码如下：

```
Console.WriteLine("连接字符串: ");
string lianjie1 = "ab";
string lianjie2 = "cd";
string lianjie3 = string.Concat(lianjie1,lianjie2);
Console.WriteLine("第一个字符串: " + lianjie1);
Console.WriteLine("第二个字符串: " + lianjie2);
Console.WriteLine("连接后的字符串: " + lianjie3);
```

(7) 获取子字符串。示例代码如下：

```
Console.WriteLine("获取子字符串: ");
string Str = "abcdef";
string ChildStr = Str.Substring(2, 2);
Console.WriteLine("原字符串: " + Str);
Console.WriteLine("子字符串: " + ChildStr);
```

(8) 查找字符串。示例代码如下：

```
Console.WriteLine("查找字符串: ");
string findstr1 = "abcd";
int index = findstr1.IndexOf('c');
Console.WriteLine("原字符串: " + findstr1);
Console.WriteLine("c 在字符串中的索引值为: " + index);
```

4. 运行结果

运行此程序，效果如图 4-14 所示。

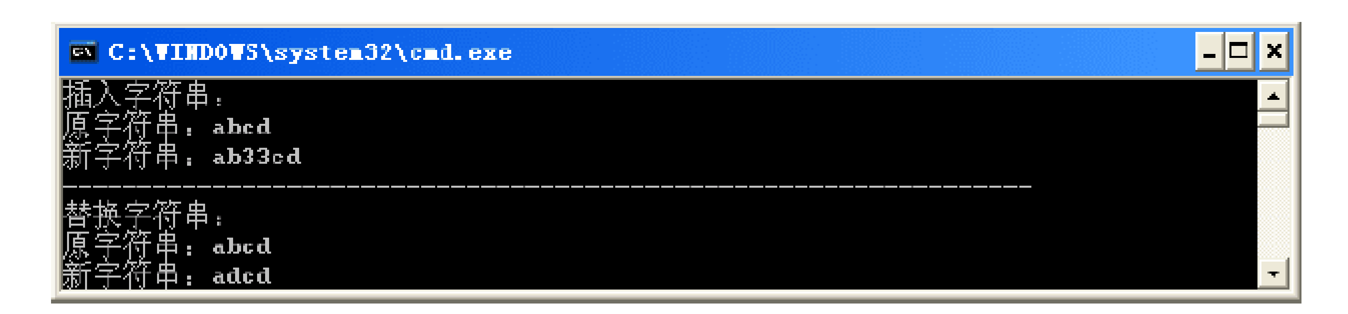


图 4-14 字符串的处理

5. 实例分析



源码解析：

该案例讲解了字符串的一些常用操作，其中比较特殊的一个操作就是分隔字符串，将字符串分隔之后，返回的是 string[]数组。数组里面的元素也是字符串，然后用 foreach 语句将这些被分隔后的字符串遍历输出。使用 Split()方法，要指定它的分隔符，分隔符一般包括任意字符。比如，分隔符是一个空格。

4.5 数据类型转换

在 Web 项目开发中，经常会遇到数据类型的转换问题，便于数据的传递及更新。类型是 C#语言中最为基本的概念之一。不同类型的变量将拥有不同的数据存储方式和操作方法。

通过本节的学习，读者将可以了解到数据类型之间的转换方法。



视频教学：光盘/videos/4/shujuleixing_zhuanhuan.avi 长度：6 分钟

4.5.1 值类型之间的数据转换

所有的值类型都直接或间接派生于 System.ValueType 类。

1. 基础知识——类型转换

值类型之间的数据是可以相互转换的。如果只是数据表示范围不同的数据之间的转换，有两种常见的转换方式：显式转换和隐式转换。如果是不同值类型之间的转换，就需要使用 Convert 类来转换。

(1) 显式转换

显式转换是指需要在数据前面添加强制转换的类型的转换。比如，声明两个变量：a 和 b。a 变量的类型是 long，值为 2010；b 变量的类型为 int，让它的值与 a 的值相等，示例代码如下：

```
long a = 2010;
int b = (int)a;
```



“b = (int)a” 中的 int 就是强制转换后的类型。

(2)　隐式转换

隐式转换是指不需要添加强制转换的类型的转换。比如，声明两个变量：a 和 b。a 变量的类型是 int，值为 2010；b 变量的类型为 long，让它的值与 a 的值相等。将 a 变量的类型隐式地转换为 b 变量的类型。

示例代码如下：

```
int a = 2010;
long b = a;
```

(3)　Convert 类

它可以将一个基本数据类型转换为另一个基本数据类型。它是一个静态的类，里面包含很多方法，这些方法大多是用来转换数据类型的，其中包含值类型之间的转换。常用的数据值类型转换方法如表 4-7 所示。

表 4-7　Convert类的常用值类型转换方法

说　明	属　性
ToInt32()	将指定的参数值转换为等效的 32 位有符号整数
ToDecimal()	将指定的参数值转换为等效的 System.Decimal 数值
ToDouble()	将指定的参数值转换为等效的双精度浮点数

2. 实例描述

程序写了老半天，终于写出来了，那就运行效果看看吧。哦，天啊，出错了，居然忘了类型转换。失误，重大失误，赶快改掉。

其实，在我们做项目的过程中，最容易忽略的就是这些小细节的问题了，但是千万别小看这些小细节，它能让你的程序运行不起来。所以，千万不要大意。

3. 实例应用

【例 4-14】值类型之间的转换。代码如下：

```
double d = 103.9000009;
int a = Convert.ToInt32(d);
Console.WriteLine(a);
```

4. 运行结果

运行此程序，效果如图 4-15 所示。



图 4-15　值类型之间的转换

5. 实例分析



源码解析：

该案例使用了 Convert 类的类型转换方式，在几种类型转换方式中，它是最常用并且不容易出错的。这里以 double 类型和 int 类型为例，将 double 类型转换为 int 类型，关键代码是 Convert.ToInt32(d)。

4.5.2 引用类型之间的数据转换

所有的引用类型都直接或间接派生于 System.Object 类。

1. 基础知识——引用类型的转换

引用类型之间的数据转换，一般包括 Convert 类的转换和强制类型转换。引用类型主要包括对象类型、字符串类型、类类型、数组类型、接口类型和委托类型。引用类型变量的值是对该类型的某个实例的一个引用。下面我们以 Convert 类的引用类型之间的转换为例，常用的方法如表 4-8 所示。

表 4-8 Convert类的常用引用类型转换方法

方　法	说　明
ToString()	将指定的参数值转换为与它等效的 System.String 表示形式
ToDateTime()	将指定的参数转换为时间表示形式

2. 实例描述

“天将降大任于斯人也”，但前提是要想成大事，必须从小事做起，这点是深有体会的。

我们在做项目的时候，往往会出现一些经常出现的问题，比如说应用程序出错、未将对象引用到实例等，其原因大多是类型转换出错。所以，这个问题很关键，也很重要。那么，下面的案例中，就会为大家讲解类型转换中的一种——引用类型之间的转换。

3. 实例应用

【例 4-15】引用类型之间的转换。代码如下：

```
string date = "2010-9-17";
DateTime date2 = Convert.ToDateTime(date);
Console.WriteLine(date2);
```

4. 运行结果

运行此程序，效果如图 4-16 所示。

```
C:\WINDOWS\system32\cmd.exe
2010-9-17 0:00:00
请按任意键继续. . .
```

图 4-16　引用类型之间的转换

5. 实例分析



源码解析：

该案例使用了 Convert 类的类型转换方式，在几种类型转换方式中，它是最常用并且不容易出错的。这里以 string 类型和 DateTime 类型为例，将 string 类型转换为 DateTime 类型，关键代码是 Convert.ToDateTime(date)。

4.5.3　装箱与拆箱

装箱和拆箱是 C#的类型系统中两个很重要的概念，它在值类型和引用类型之间架起了一座桥梁。通过装箱和拆箱操作，可以将任何值类型的变量的值转换为引用类型的变量的值。反之，也可以进行转换。特别是，有了装箱和拆箱操作，就可以使 C#类型系统中的任何类型的值最终都可以按对象来处理。

1. 基础知识——装箱与拆箱概念

将值类型转换为引用类型，被称为装箱。反之，将引用类型转换为值类型，被称为拆箱。



被装过箱的对象才能被拆箱。

(1) 装箱

装箱是指将值类型转换为引用类型的过程。对于值类型来说，装箱的具体操作为：首先分配一个对象实例，然后将值类型的值复制到该实例中。装箱前后不是同一个实例。对于引用类型来说，装箱前后都是同一个实例。

(2) 拆箱

拆箱是指将引用类型转换为值类型的过程。拆箱之前，要先检查该对象实例是否为给定值类型的一个装过箱的值，然后将值从实例中赋值出来。

2. 实例描述

中秋节快到了，月饼也就卖得很贵了，生产厂家必须将月饼经过包装，才会卖出更高的价格。所以消费者就必须把那层华丽的套装脱掉，才能享用美食。

装箱与拆箱也是同样的道理，下面就为大家讲解一下所谓的“装箱”与“拆箱”吧。

3. 实例应用

【例 4-16】装箱与拆箱。代码如下：

```
Console.WriteLine("一个整型数值为：2010");
int i = 2010;
object obj = i;
Console.WriteLine("装箱后为：" + obj);
int j = (int)obj;
Console.WriteLine("拆箱后为：" + j);
```

4. 运行结果

运行此程序，效果如图 4-17 所示。



图 4-17　装箱与拆箱

5. 实例分析



源码解析：

装箱与拆箱就是在值类型与引用类型之间进行转换。其中，int 类型与 object 类型之间的转换，就是装箱与拆箱的原型。

声明一个整型变量 i，它的值为 2010，装箱是声明一个 object 对象 obj，然后使“obj=i”；拆箱与之相反，声明一个整型变量 j，那么“j=(int)obj”。

4.6　操作学生信息实体——基于类的泛型

泛型(Generic)是具有占位符(类型参数)的类、结构、接口和方法，这些占位符是类、结构、接口和方法所存储或使用的一个或多个类型的占位符。

使用泛型类型可以最大限度地重用代码、保护类型的安全及提高性能。泛型最常见的用途是创建集合类。.NET Framework 类库在 System.Collections.Generic 命名空间中包含多个新的泛型集合类，如 List<T>、Stack<T>、Queue<T>等。当然，开发人员也可以创建自定义泛型类型。



视频教学：光盘/videos/4/fanxing.avi　　　　长度：2 分钟

4.6.1　基础知识——泛型

我们来操作学生信息实体。这里的学生信息实体表示一个保存学生信息的类。所以声明一

个类，语法如下：

```
public class 类名
```

然后实例化一个泛型类，语法如下：

```
List<类名> lst = new List<类名>();
```

List<类名>常用的几个方法如表 4-9 所示。

表 4-9　List<类名>的常用方法

方　法	说　明
Add()	将对象添加到 System.Collections.Generic.List<T>的结尾处
Clear()	从 System.Collections.Generic.List<T>中移除所有元素
Insert()	将元素插入 System.Collections.Generic.List<T>的指定索引处
Remove()	移除与指定的谓词所定义的条件相匹配的所有元素

4.6.2　实例描述

最近，一个同事询问用泛型保存数据怎样写，刚巧，把作者给问住了。作者一般都是直接从数据库里读取或者从 XML 文件读取。于是，就从网上了解了一下泛型的用法，觉得用泛型保存数据也挺方便的，而且不会占用数据库的连接资源，所以想给读者们推荐一下这个保存数据的方法。

4.6.3　实例应用

【例 4-17】操作学生信息实体——基于类的泛型。

(1)　创建一个学生类，名为 stu，包含 stuid 和 name 两个字段。示例代码如下：

```
public class stu
{
    /// <summary>
    /// 学生编号
    /// </summary>
    public int stuid { get; set; }
    /// <summary>
    /// 学生姓名
    /// </summary>
    public string name { get; set; }
}
```

(2)　然后在控制台程序的 Main 函数中调用。实例代码如下：

```
List<stu> lst = new List<stu>();
lst.Add(new stu { stuid=1,name="李小刚" });
lst.Add(new stu { stuid=2,name="方一" });
```

```
foreach (stu s in lst)
{
    Console.WriteLine(s.stuid + "    " + s.name);
}
```

4.6.4 运行结果

运行此程序，效果如图 4-18 所示。



图 4-18 操作学生信息实体

4.6.5 实例分析



源码解析：

学生类里的两个字段 stuid 和 name 分别是 int 类型和 string 类型。其中{get; set;}访问器是.NET Framework 3.5 版新增的一种形式。用 Add()方法添加学生信息，要先实例化学生信息，它表示一个对象。例如：

```
new stu { stuid=1, name="李小刚" }
```

最后将泛型里的对象的 stuid 和 name 值用 foreach 语句遍历输出。

4.7 常见问题解答

4.7.1 C#中的隐式转换问题



C#中的隐式转换问题。
网络课堂：http://bbs.itzcn.com/thread-9856-1-1.html

为什么写 float x1=3.0 时会提示：不能将 double 类型隐式转换成 float，而写成 float x1=3 或 float x1=3.0F 却可以编译呢？

【解决办法】：嗯……是这样的，因为在 float x1=3.0 中，你没有在“3.0”后面加一个 f 或者 F，所以系统就会默认为“3.0”是一个 double 类型的数据，所以就不能隐式转换了。f 或 F 是代表 float 的意思，加上 f 后，3.0f 就表示是 float 类型。

而对于所说的 float x1=3 可以编译，是因为“3”也没有加 f 或 F，所以系统会默认“3”

为 int 类型，而 int 类型可以隐式转换为 float 类型，所以当然就可以编译了。

另外，需要说明的是，从 double 转换到 float 会造成精度丢失，所以要显式转换才行。可以这样转换：float f=(float)3.0。

4.7.2　C#数组问题



C#数组问题。
网络地址：http://bbs.itzcn.com/thread-2599-1-1.html

在某书上看到：

```
int arrayLenth = 10;
int []intArray = new int[arrayLenth];
```

arrayLenth 不是常量？这样不就可以改变数组的大小了吗？

【解决办法】：说得对，arrayLenth 的确不是常量，它是一个变量，但是你不能改变数组的大小。可以这样定义数组：在你还没有声明数组之前可以改变数组的大小，但声明之后，数组的大小是不能改变的。如果你需要创建动态数组，可以用 ArrayList，或其他的集合。ArrayList 类可以通过添加或删除数组元素，从而动态地改变数组的长度。

4.8　习　　题

一、填空题

(1) 数组的下标从__________开始。
(2) 集合的命名空间是__________。
(3) 获取当前完整的日期和时间，语句是__________。
(4) 向文章里插入一段文字，用 String 类的__________方法。
(5) 装箱是将值类型转换为__________。折箱是将__________转换为值类型。

二、选择题

(1) 遍历数组可以使用以下________循环语句。
A．for　　B．while　　C．do...while　　D．foreach
(2) 下列________选项不是格式化时间的方法。
A．ToLongDateString()　　B．ToShortDateString()
C．ToDateTime()　　D．ToString()
(3) 下面________选项用来替换字符串。
A．Remove()　　B．Replace()　　C．Insert()　　D．Append()
(4) 可变字符串的类是________。
A．string　　B．String　　C．Stringbuilder　　D．StringBuilder
(5) 将 int i=100;中的 i 装箱，下列选项正确的是________。

A．object obj=i　　B．double db=Convert.ToDouble(i)

C．long g=(long)i　　D．string str=Convert.ToString(i)

三、上机练习

上机练习 1：获取当前日期，并格式化为年月日的形式。

首先获取当前日期，然后将它以年月日的形式显示。效果如图 4-19 所示。

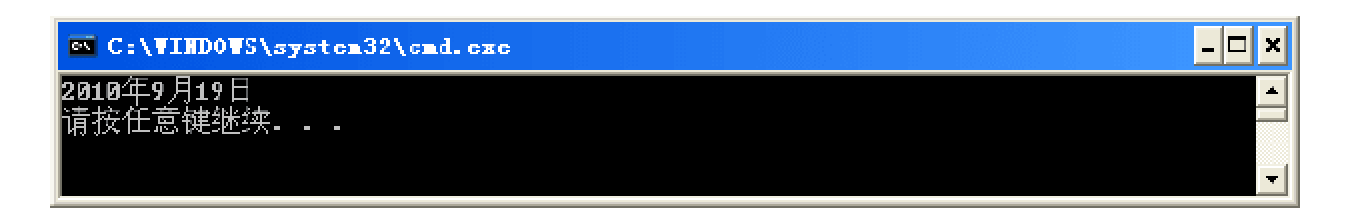


图 4-19　日期转换

上机练习 2：遍历输出字符串数组。

声明一个名为 ArrStr 的字符串数组，初始化数组的值分别为“Hello”，“World”，“Nice”。最后遍历该数组的每一个元素。效果如图 4-20 所示。



图 4-20　遍历输出字符串数组

上机练习 3：向集合列表里添加元素。

实例化一个名为 score 的集合列表，列表类型为 double，然后遍历该列表中的元素，效果如图 4-21 所示。



图 4-21　向集合列表里添加元素



第 5 章　构建永久的信息仓库

内容摘要：

现在的应用软件越来越多了。除了一些工具软件之外，几乎所有的软件系统都要收集和存储信息。数据信息要持久保存，一般需要将信息以文件的形式保存到磁盘等存储介质上。存储数据一般要用到的是数据库技术，也可以用直接操作文件的形式进行保存。

数据库技术从 20 世纪 60 年代起到现在已经发展了将近半个世纪，优秀的数据库技术也是层出不穷。当下最为流行的数据库莫过于甲骨文公司的 Oracle、微软的 SQL Server、MySQL AB 的 MySQL、微软公司的 Access。当然还有 IBM 的 DB2、Sybase 公司的 Sybase 等。

因为每个数据库都有各自的优缺点，所以对各种数据库的访问也不尽相同。不过有幸的是它们都遵循了一个标准——T-SQL，使我们学习和使用起来减少了难度。.NET Framework 提供了一系列的类库，实现了对各种数据库的操作，并把它们通称为 ADO.NET。

如果是非常少量的数据，在数据库里建表存储就显得有点过于费事了。以前我们的做法是自己定义格式，以文本的形式保存到文件中。不过.NET Framework 提供了一个序列化机制，可以直接把对象以二进制的形式保存，方便至极。

在本章，我们就来了解一下 ADO.NET 提供的数据库访问机制和可以直接存储数据对象的序列化功能。

学习目标：

- 了解 ADO.NET 的工作原理
- 掌握 SqlCommand 对象的用法
- 掌握 SqlDataReader 对象的用法
- 熟练使用 SqlParameter 对象给 SqlCommand 对象传参
- 掌握 SqlDataAdapter 和 DataSet 对象的用法
- 熟练使用 ORM 框架 Linq to SQL
- 掌握对象序列化技术

5.1　建造到各个数据库仓库之间的桥梁

一般应用程序为了有更好的扩展性和可移植性，通常把数据库独立出来单独存放或运行。

其实这样做还能提高程序的运行性能、程序数据的安全性。因为数据库可以单独存储在一台服务器上，减轻了应用程序服务器的负担。同时在应用程序出现问题时不能直接影响到数据库数据。当然，这样使用独立数据库服务器的方法等于在网络上公开了数据库。为了保证数据的安全性，数据库服务器也设置了层层障碍，来阻挡非法用户的入侵。

然而，戒备森严的数据库服务器也给所有客户提供一种比较安全的访问方式，那就是 Connection。

就像一座湖心孤岛，周围有湖水作为天然屏障环绕保护，岛上防空炮无数，四周地雷、水雷 N 多。总之水陆空三军层层防守，想要上岛办事，必须在对应岛上指定的地方先修一座指定规格的桥才行。

当然，桥是谁想修就修了，但是桥头也有荷枪实弹的守卫看门，必须持有通行证或者对上口令后才能通过，否则桥修了也白修。

在这个领域里，提供安保服务的组织也相当多，有 SQL Server，有 MySQL，有 Oracle。你修的桥若不合规定，肯定也不让你过了，说不定就在桥头布设了一片雷区。有非法过桥时，一个 Error 就给你炸回来了。

为了不被“炸”死，下面我们来认识一下这些强大的组织。

5.1.1　连接到SQL Server酒店管理数据库

SQL Server 是软件帝国 Microsoft 旗下的“安保公司”，业务范围广，功能强大。我们这次请它来管理和看护我们的数据仓库。

早就听说安保较真时是六亲不认的，所以接下来我们了解一下 SQL Server 的访问方式。我们建立一个 Connection 试试。

 **视频教学：** 光盘/videos/5/ConnectDb.avi　　　　**长度：** 9 分钟

1. 基础知识——SqlConnection

ADO.NET 提供了一套类库，专门来访问 SQL Server 数据库。都在 System.Data.SqlClient 命名空间下面。

该命名空间提供了访问 SQL Server 数据库所有的类。包括最常用的 SqlConnection、SqlCommand、SqlDataReader、SqlDataAdapter 等。

创建 SQL Server 数据库 Connection 时要用到 SqlConnection 类，它提供如下两个构造方法。

- SqlConnection()：创建一个 SqlConnection 对象。
- SqlConnection(string connectionString)：创建一个 SqlConnection 对象，并且初始化连接字符串非常简单。

使用第一个方法，就可以创建一个完整的桥，需要使用的时候直接将桥的一头搭在目标地点上就行了，就是前面我们说的岛上的指定地点。而第二个构造方法就指定了桥那一头要搭向的目标地点和搭桥的通行证 connectionString。

第一个方法没有指定目标地址，不过不用担心，SqlConnection 类还提供了一个属性 ConnectionString，来指定连接字符串，用来在创建 SqlConnection 对象后初始化连接字符串。

前面提到了连接字符串 connectionString。看似简单，就一个字符串，其内容相当复杂。目标地址、目标数据库、访问账号、访问密码等信息它都指定了。下面来看个例子：

```
Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123
```

我们来分析它的结构。

- Data Source：看字面意思是“数据来源”，就是指数据库在哪里存放着，是一个主机地址。这里是作者的计算机名称
- Initial Catalog：初始目录，就是数据库。也就是说这个岛上住了很多人，你要去找谁，你得说清楚，否则一样不让你过。
- User ID：用户 ID，区分用户的唯一编号。某些用户你只能来看看，不能带任何东西进来，有些用户你只能来办你的事，不能乱看。这些差别都由它来区分。
- Password：当然，这就是登录口令了，对不上口令就一脚踢回去了。

还有一种方式是使用 Windows 验证方式，实际应用得不多，就不介绍了。

回到 SqlConnection，创建完对象以后，使用的时候需要打开对象。就像上面的例子说的那样，你建造了一个活动的桥，当然要搭在目标地点上才能使用。

SqlConnection 对象的打开方法非常简单，调用 Open()方法即可。例如：

```
conn.Open();
```

这里的 conn 是 SqlConnection 类的实例。

当然了，修桥是要占地方的。岛再大可以修桥的地方也是有限的。所以修好桥，办完事，拉完东西，还需要赶紧把桥拆了给别人腾地方。

同样使用 Connection 建立连接也是浪费资源的，用完就关是聪明的，用完不关就有点不聪明了。

关闭的方法很简单。例如：

```
conn.Close();
```

下面我们就来连接一下数据库试试。

2. 实例描述

不久前我们接到一个项目，开发一个酒店管理系统，客户提出要使用 SQL Server 2005 作为数据库。

客户既然有要求，我们就执行吧，因为客户是上帝啊。

以前没用过 ADO.NET + SQL Server，不过凭我们的聪明智商，肯定是非常容易的事情。

当然我们首先得研究一下怎么连接吧，如果数据库都连不上，一切伟大的决定都是废话。

SqlConnection 还提供了一个 State 属性来返回连接状态。我们就在对 SqlConnection 对象每操作一次时向屏幕打印一次连接状态。

3. 实例应用

【例 5-1】 连接到 SQL Server 的酒店管理数据库。

(1) 首先我们得建一个测试项目，这里就用控制台程序测试连接，它省事、代码少、干净，运行也快速。还有一个更重要的原因是，很多初学者都对黑乎乎的屏幕有一种莫名的神秘感，所以我们也借此机会炫耀一下。

(2) 修改应用程序的 Main 方法，添加代码如下：

```
static void Main(string []args)
{
    //创建连接字符串
    string connectionString =
        "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";
    SqlConnection conn = new SqlConnection();    //初始化连接对象
    Console.WriteLine("创建对象后：" + conn.State);    //打印连接状态

    conn.ConnectionString = connectionString;    //初始化连接字符串
    Console.WriteLine("初始化链接字符串后" + conn.State);    //打印连接状态

    conn.Open();    //打开连接
    Console.WriteLine("打开连接后：" + conn.State);    //打印连接状态

    conn.Close();    //关闭连接
    Console.WriteLine("关闭连接后：" + conn.State);    //打印连接状态

    Console.ReadLine();    //程序暂停，回车继续
}
```

完成了，就这么简单。

4. 运行结果

按 F5 键或单击工具栏中的“启动调试”按钮来运行程序，结果如图 5-1 所示。

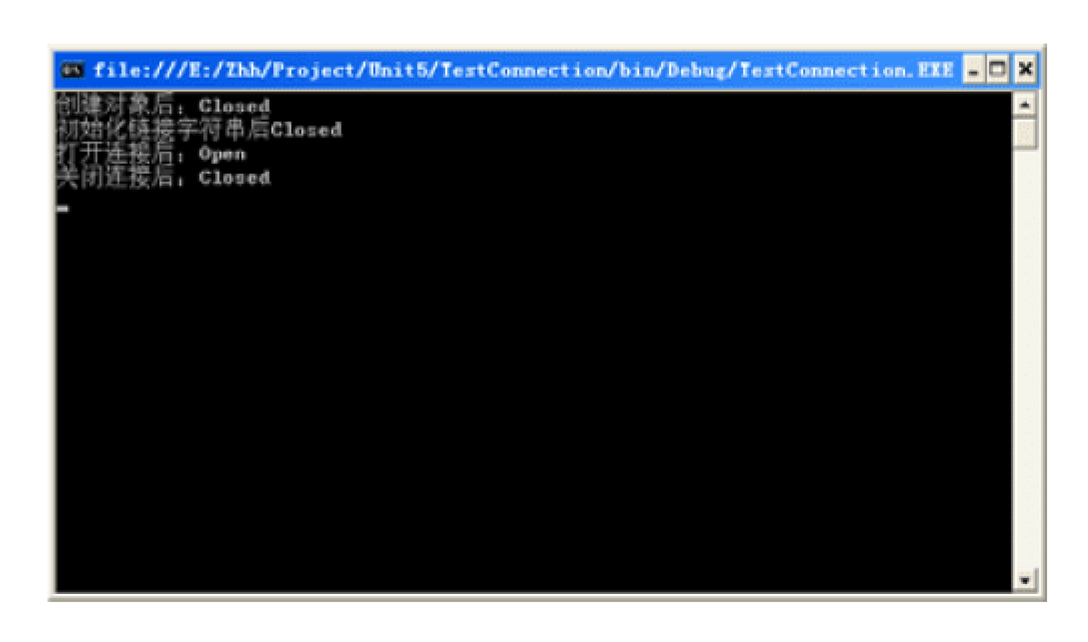


图 5-1 测试SqlConnection(1)

我们看到，只有在 SqlConnection 对象执行过 Open()方法以后，在执行 Close()方法以前，SqlConnection 对象的状态才是打开的。

整个实例就讲完了，不过过程有点过于顺利了吧。

假设连接字符串错了，是什么情况呢？

我们来修改一个连接字符串的账号或密码，改成一个错的信息再来运行。

注意这里不能再按 F5 键或单击“启动调试”按钮了，那样调试的时候会停在出异常的语句上，所以这里按 Ctrl+F5 组合键不调试直接启动和运行项目。

运行结果如图 5-2 所示。

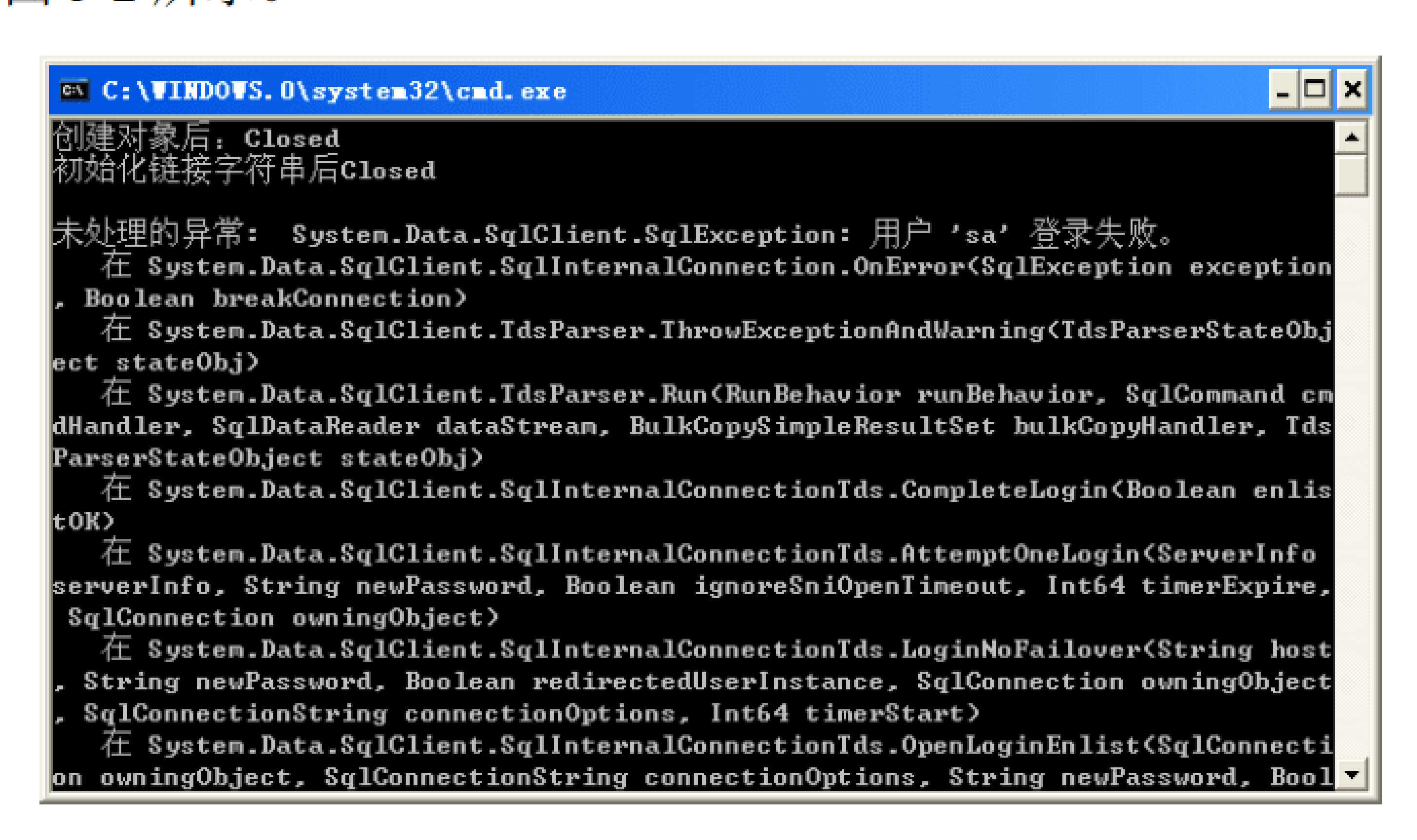


图 5-2　测试SqlConnection(2)

可以看到，在创建 SqlConnection 对象的时候和初始化连接字符串的时候，还正常地打印了 SqlConnection 对象的状态，在执行 Open()操作的时候就不行了，报了一个用户 sa 登录失败。说明我们试图非法请求 SQL 服务器时被踢回来了。

5. 实例分析



源码解析：

该实例编辑了控制台程序的程序入口点 Main 方法。先创建了一个声明数据库连接信息的连接字符串，然后使用 SqlConnection 类的默认构造方法创建一个 SQL 数据库连接对象，然后初始化该对象的 ConnectionString 属性。然后调用 SqlConnection 类的 Open 方法打开连接，最后关闭连接释放资源。为了监控其状态，每次对 SqlConnection 对象执行操作后就向控制台打印一下该对象的状态。

5.1.2　基于ODBC数据源连接的手机订单数据库

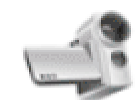
下面来了解一下什么是 ODBC。

ODBC 代表开放数据库连接(Open DataBase Connectivity)，是微软公司开放服务结构(Windows Open Services Architecture，WOSA)中关于数据库的一个组成部分。它建立了一组规范，并提供了一组访问数据库的标准 API，这些 API 独立于不同厂商的 DBMS，也独立于具体的编程语言。

当然上面是很多资料上对它的总结，听起来晕晕乎乎的，根本听不明白 ODBC 到底是什么样的东西。

试了试，才知道它原来是一个适配器一样的东西，应用程序访问它，然后它再访问其他数

据库，只是像中间代理了一下一样。这样如果要更换应用程序数据库的话，就不用修改应用程序了，因为应用程序只需要访问 ODBC，至于最终是哪种数据库，应用程序不用过多关心。


视频教学：光盘/videos/5/ConnectionStr.avi　　长度：8 分钟

1. 基础知识——OdbcConnection

ADO.NET 也提供了一套专门用来访问 ODBC 的类库。都在 System.Data.Odbc 命名空间的下面。同样，它也有相对应的操作数据库的类。例如 OdbcConnection、OdbcCommand、OdbcDataReader、OdbcAdapter 等。

与 SqlConnection 一样，OdbcConnection 也提供了两个构造方法。一个是无参的，一个是带数据库连接字符串初始化的。这里就不再解释。

我们来看它的连接字符串：

```
Dsn=MobileOrder
```

Dsn 是 Data Source Name。

很简单，不多说了。


这里连接的是作者本机的 ODBC，所以可以不用账号和密码直接访问。如果要访问其他计算机的 ODBC，还需要设置账号和密码。

2. 实例描述

从来没有用过 ODBC，确实它的性能不怎么好。而且对于数据库的更换，现在的应用程序已经做了大量的考虑，数据库偶尔的更替也不怎么令人困惑发狂了。

所以 ODBC 现在在市场上出现的频率非常低。

假如 ODBC 是另一家不知名的公司开发和推广的，肯定早就已经退出市场了吧。

一次偶然相见，是一个朋友打电话说他维护的一个电信运营商的公用电话系统出了故障，让我见识了一下使用 ODBC 的应用系统。

那个系统是 2000 年左右的产品，很古老，十来年了，也早该淘汰了，无奈领导却想凑合着运行……

自从那次见了以后，回来就用 C#测试了一下这个传说中的东西。

3. 实例应用

【例 5-2】基于 ODBC 数据源连接的手机订单数据库。

前面说过，ODBC 是一个数据库的代理。所以要建立一个对已存在数据库的数据源。

作者用的是 Windows XP 系统，在“控制面板”里打开“管理工具”，“管理工具”里有一个“数据源(ODBC)”图标，运行它就是了。

在这里可以添加一个“数据源”，这里的数据源还是比较全的：Access、Excel、dBase、FoxPro、SQL Server，你想用哪种就用哪种。

添加一个以后，在列表里会看到你刚添加的那个数据源，这里添加了一个以前做的一个手机订单程序的数据库，数据源名称是 MobileOrder。

好了，运行上节我们的项目。修改 Main 方法，如下所示：

```
static void Main(string []args)
{
    OdbcConnection conn =
      new OdbcConnection("Dsn=MobileOrder"); //初始化连接对象
    Console.WriteLine("创建对象后: " + conn.State);    //打印连接状态

    conn.Open();    //打开连接
    Console.WriteLine("打开连接后: " + conn.State);    //打印连接状态

    conn.Close();    //关闭连接
    Console.WriteLine("关闭连接后: " + conn.State);    //打印连接状态

    Console.ReadLine();   //程序暂停，回车继续
}
```

当然，前面需要引入这个实例所需要的命名空间：System.Data.Odbc。

程序代码编写完毕，我们来运行它。

4. 运行结果

运行以后会弹出一个控制台窗口，打印出程序执行结果，如图 5-3 所示。

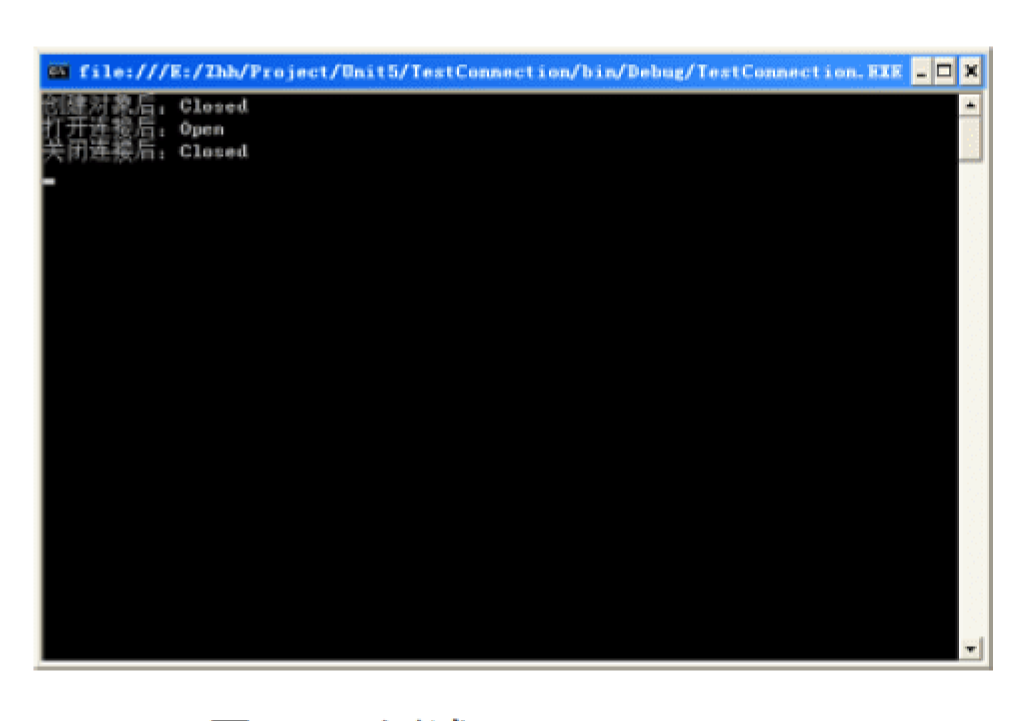


图 5-3　测试OdbcConnection

5. 实例分析



源码解析：

实例在应用程序入口 Main 方法里调用 OdbcConnection 类的带一个连接字符串参数的构造方法，创建了一个 OdbcConnection 对象，创建该对象的同时已经初始化了连接字符串。接下来用 OdbcConnection 对象的 Open 方法打开数据连接，然后调用 Close 方法关闭连接并释放资源。在执行创建对象、打开连接、关闭连接的操作后打印连接对象的状态。

5.1.3　连接个人博客Access数据库

Microsoft Office Access(前名 Microsoft Access)是由微软发布的关联式数据库管理系统。它结合了 Microsoft Jet Database Engine 和图形用户界面两项特点，是 Microsoft Office 的成员之一。

Access 是一般单机小应用程序的首选。它功能强大，效率也相当高。最吸引人的还是它的可移植性，它只是一个文件，不用服务器，也不用额外运行什么程序驱动它，使用非常方便。

这里说的“不用额外运行什么程序驱动它”可别理解成不用程序驱动它。因为它是微软开发的，而微软总想在软件行业搞点垄断，于是就直接把 Jet 数据库引擎(Jet Database Engine)集成到 Windows 系统里了，所以可以直接使用 Access 数据库。



Access 存储和处理小量数据时的确性能十分优异。不过有人测试过，在 Access 数据库容量超过 30MB 左右的时候就几乎没法用了，超慢！


视频教学：光盘/videos/5/ConnectionStr.avi　　长度：8 分钟

1. 基础知识——OleDbConnection

同样是 Microsoft 的产品，ADO.NET 当然对 Access 数据库提供了很好的支持。

Access 可以被基于 COM 的数据存储对象 OleDb 访问，而在 ADO.NET 里提供了一组专门用来访问 OleDb 的类。都在 System.Data.OleDb 命名空间下面。

这一组类中同样有几个基本的类 OleDbConnection、OleDbCommand、OleDbDataReader、OleDbDataAdapter。OleDbConnection 的使用方法同样和 SqlConnection 一样。

我们来看它的连接字符串：

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=MyBlog.mdb
```

这里有两个参数。

- Provider：数据提供程序，这里就是我们连接 Access 需要的驱动名称和版本。
- Data Source：数据源，这里是我们的 Access 数据库文件地址。

当然，如果数据库有密码，还需要追加上账号和密码。不过 Access 密码很容易破解，有和没有差不太多，这里就不使用了。

2. 实例描述

网上提供的博客空间模版太固定，没有一点新意，没有一点个性。所以就自己做一个博客，并弄个域名空间发布。

当然，由于不是什么名人，所以访问量不高，数据量也不大，就首选 Access 作为个人博客程序的数据库。

既然这里要讲解以 C#连接 Access 数据库，就把作者站点的 DAL 层连接数据库代码取出来演示一下。

3. 实例应用

【例 5-3】连接个人博客 Access 数据库。

运行上节我们的项目。修改 Main 方法如下：

```
static void Main(string []args)
{
    //创建连接字符串
    string connStr =
      "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=MyBlog.mdb";
```

```
    OleDbConnection conn = new OleDbConnection(connStr);    //初始化连接对象
    Console.WriteLine("创建对象后: " + conn.State);    //打印连接状态

    conn.Open();    //打开连接
    Console.WriteLine("打开连接后: " + conn.State);    //打印连接状态

    conn.Close();    //关闭连接
    Console.WriteLine("关闭连接后: " + conn.State);    //打印连接状态

    Console.ReadLine();    //程序暂停，回车继续
}
```

4. 运行结果

运行后会弹出一个控制台窗口，打印出程序执行结果，如图 5-4 所示。

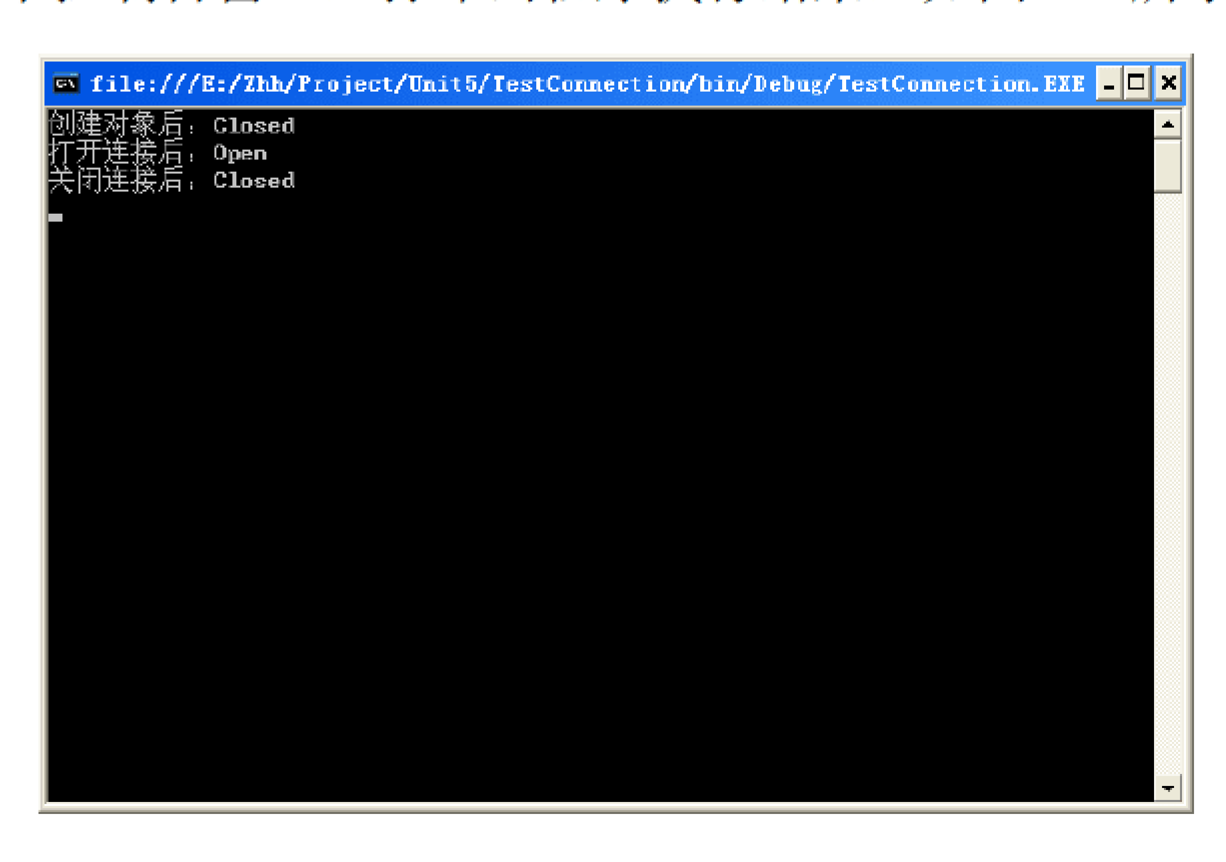


图 5-4　测试OleDbConnection

5. 实例分析



源码解析：

由于连接 Access 数据库的连接字符串比较冗长，所以这里单独声明一个 string 类型的连接字符串，然后使用该连接字符串作为参数创建并初始化 OleDbConnection 对象，此时我们打印一下数据库连接对象的状态。接着调用该对象的 Open 方法打开数据库连接，然后再次打印连接对象的状态。然后调用该对象的 Close 方法关闭该对象并释放资源。

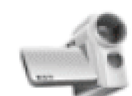
5.1.4　连接到远程房产交易Oracle数据库

Oracle，多少年来，其专注专业的精神使它一直是数据库领域里的王者。

在国内，只要是稍大一点的项目，几乎都是使用 Oracle 数据库，大量大型用户、公司的支持，也帮助捍卫了它王者的地位。

前段时间它收购了 Sun，不知道会不会像微软一样稍稍施加一点垄断手段，让 Java 的拥护

者也死心塌地地拥护 Oracle。

 **视频教学：** 光盘/videos/5/ConnectionStr.avi　　长度：8 分钟

1. 基础知识——OracleConnection

前面说了 Oracle 很有实力。而现实是尊重实力的，就连同样有实力的 Microsoft 也不敢冒天下之大不韪在自己的产品里不支持它。

所以 ADO.NET 提供了一套专门用来访问 Oracle 数据库的类，在 System.Data.OracleClient 命名空间下面。

同样，它也具有类似的那几个类：OracleConnection、OracleCommand、OracleDataReader、OracleDataAdapter。

OracleConnection 类同样有那两个构造方法，同样有打开和关闭连接的类。

我们来看它的连接字符串：

```
data source=house;user id=scott;password=tiger
```

这里使用了 3 个参数。

- data source：数据源，就是 Oracle 里的"网络服务名"。
- user id：登录账号。
- password：登录密码。

2. 实例描述

去年作者公司的另一个项目组做过一个房产交易系统，用的是 Oracle 数据库。这里为了演示，就把数据库拉到作者的机器上连接一下。

3. 实例应用

【例 5-4】连接到远程房产交易 Oracle 数据库。

运行上节我们的项目。修改 Main 方法如下：

```
static void Main(string []args)
{
    string connStr = "data source=house;user id=scott;password=tiger";

    OracleConnection conn = new OracleConnection(connStr);
    Console.WriteLine("创建对象后：" + conn.State);    //打印连接状态

    conn.Open();    //打开连接
    Console.WriteLine("打开连接后：" + conn.State);    //打印连接状态

    conn.Close();    //关闭连接
    Console.WriteLine("关闭连接后：" + conn.State);    //打印连接状态

    Console.ReadLine();    //程序暂停，回车继续
}
```

4. 运行结果

运行后会弹出一个控制台窗口，同样打印出程序执行结果，如图 5-5 所示。

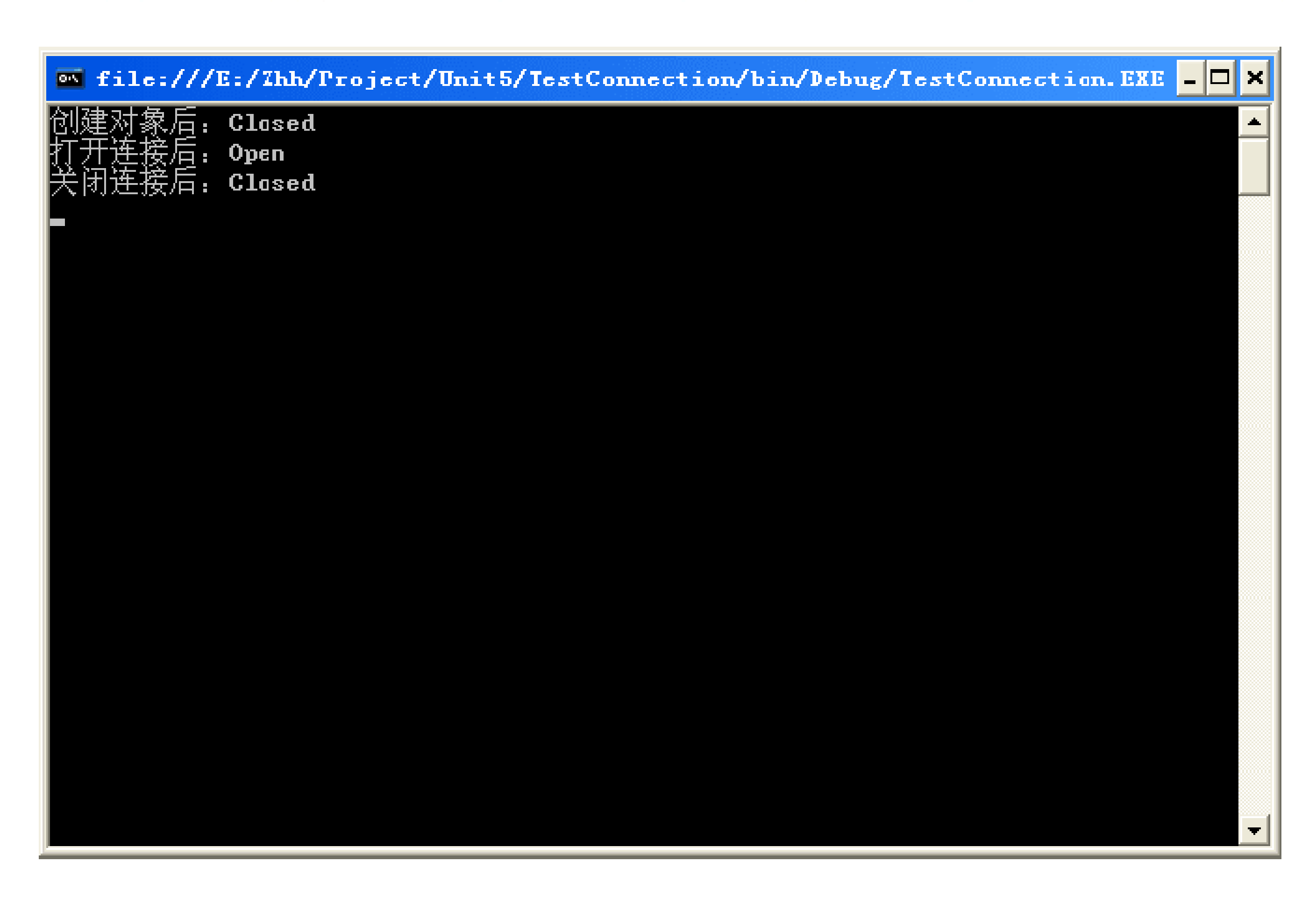


图 5-5　测试OracleConnection

5. 实例分析



源码解析：

实例先初始化一个 string 类型的 Oracle 数据库连接字符串 connStr，然后使用该连接字符串创建并初始化一个 OracleConnection 对象，并打印出创建并初始化以后该对象的状态。接下来我们调用 OracleConnection 类的 Open 方法打开数据库连接，再打印一下该对象的状态，然后调用 Close 方法关闭连接，同时再打印一下连接对象的状态。

5.1.5　打通与MySQL社区系统的连接

MySQL 是一个小型的关系型数据库管理系统。

其特点是体积小、速度快、免费、开源。尤其是免费这一特点，许多中小型网站为了降低成本而选择了 MySQL 作为网站数据库。

在国外，因为免费的原因，PHP 相当火热。而 PHP 和 MySQL 应该是个注定的绝配，所以在 PHP 系统上 MySQL 数据库几乎(甚至可以说已经)处于绝对垄断的地位了。

因为 PHP 和 MySQL 在一些系统上的成就，使得 MySQL 在数据库领域里也占有了不可或缺的一席之地。



视频教学：光盘/videos/5/ConnectionStr.avi　　　　长度：8 分钟

1. 基础知识——MySQLConnection

ADO.NET 并没有提供对 MySQL 数据库的支持。所以我们在使用的时候需要使用一个第三方插件 MySQLDriverCS。

下载完以后是一个可执行程序，运行并安装它。

安装完以后，应用程序还需要引用它提供的一个类库 MySQLDriverCS.dll。在安装目录下的 dll 目录下可以找到该文件。它同样提供了一些与 ADO.NET 提供的其他数据库的类库差不多一样的类：MySQLConnection、MySQLCommand、MySQLDataReader、MySQLDataAdapter 等。至于连接字符串，它还提供了一个专门生成连接字符串的类 MySQLConnectionString。使用的时候在构造方法里传入 MySQL 服务器地址、端口号、数据库名、登录账号、密码之类的信息初始化该对象，然后可以使用 AsString 属性得到一个连接字符串。

具体这里不再多说，接下来看实例。

2. 实例描述

前段时间接了一个小项目。客户要对现有的一个 PHP 系统里的一些信息做一下统计。

看了一下系统，是一个 PHP 写的社区系统，听说功能很强大。看它的数据库，是 MySQL 的。当时想，现学 PHP 有点太费时，就直接用 ASP.NET 连接 MySQL 数据库写了一个简单的统计系统，也算满足了客户的要求。

3. 实例应用

【例 5-5】打通与 MySQL 社区系统的连接。

(1) 下载安装完 MySQLDriverCS 以后。运行上节我们的项目，先对项目添加对类库文件 MySQLDriverCS.dll 的引用。

(2) 再在类文件里对添加对类库命名空间 MySQLDriverCS 的引用。

(3) 然后修改我们的 Main 方法，如下所示：

```
static void Main(string []args)
{
    //用 MySQL 连接字符串对象生成连接字符串
    string connStr = new MySQLConnectionString(
      "localhost", "ucenter", "root", "123456", 3306).AsString;
    Console.WriteLine(connStr);    //打印连接字符串

    MySQLConnection conn = new MySQLConnection(connStr);    //初始化连接对象
    Console.WriteLine("创建对象后：" + conn.State);    //打印连接状态

    conn.Open();    //打开连接
    Console.WriteLine("打开连接后：" + conn.State);    //打印连接状态

    conn.Close();    //关闭连接
    Console.WriteLine("关闭连接后：" + conn.State);    //打印连接状态

    Console.ReadLine();    //程序暂停，回车继续
}
```

这里我们生成连接字符串以后打印了，也算了解一下 MySQL 的连接字符串。

接下来运行一下看看效果。

4. 运行结果

运行以后会弹出一个控制台窗口，打印出程序执行的结果，如图 5-6 所示。

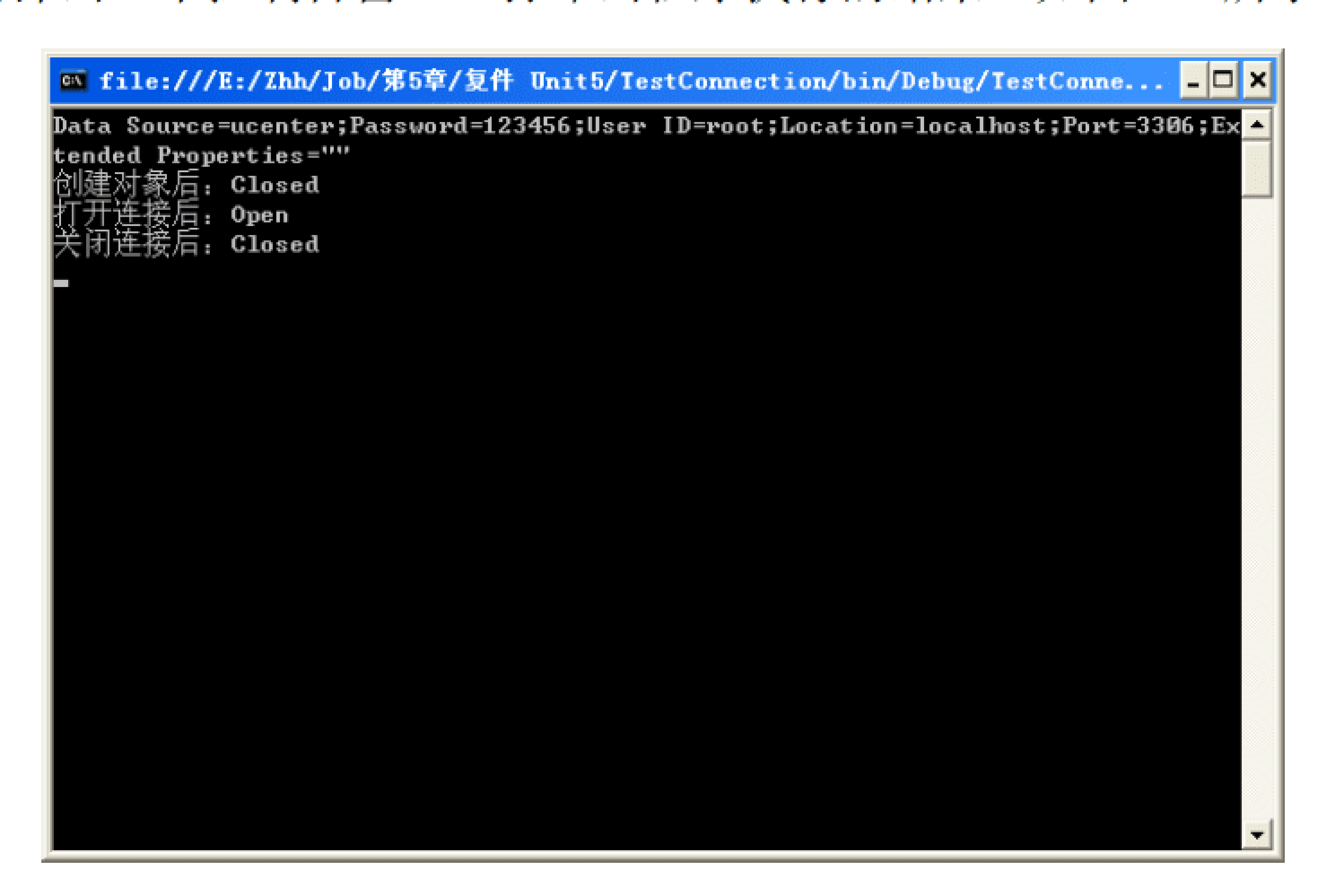


图 5-6　测试MySQLConnection

5. 实例分析



源码解析：

实例需要先下载和加载 MySQL 数据库操作支持类库 MySQLDriverCS。然后我们修改应用程序的 Main 方法。在 Main 方法里使用 MySQL 数据库连接字符串类 MySQLConnectionString 的实例生成一个数据库连接字符串，并打印出该字符串。接下来使用带参的 MySQLConnection 类的构造方法创建并初始化一个 MySQLConnection 对象，并打印其状态。然后调用 Open 方法打开连接，打印状态。然后关闭连接，释放资源，再打印连接对象的状态。

5.2　添加酒店会员

刚才连通了应用程序到数据库之间的桥梁，接下来要开始干活了。上面说了好几种数据库连接方式，我们到底选择那种数据库呢？

因为它们都支持标准的 T-SQL 语言，所以对数据库的基本操作已经没有多大差别，具体选择使用哪种数据库已经不是什么太大的问题了。这里我们就使用 SQL Server 2005 来作为我们的数据库服务器。

有了我们上面修的桥梁，怎样把我们的数据送到目的地址呢？

接下来就用到无微不至的助理 SqlCommand。



视频教学：光盘/videos/5/SqlCommand.avi　　　　长度：13 分钟

5.2.1 基础知识——SqlCommand

前面说了，SqlCommand 是一个助理，这确实很形象。它的工作就是接受应用程序操作数据库的命令，奔赴在数据库和应用程序之间。存数据、取数据。也可以说是一个仓库调度员。

SqlCommand 继承自 DbCommand。它重载了几个构造方法，用以初始化命令文本、使用的连接对象和所属的事务。当然，还有一种创建 SqlCommand 对象的方法，就是使用 SqlConnection 对象的 CreateCommand()方法，这个比较常用。

有几个不可忽视的属性需要我们注意一下。

- CommandText：SqlCommand 对象需要执行的命令文本。可以是 T-SQL 语句、存储过程等。
- CommandType：SqlCommand 对象的命令类型。值为 CommandType 枚举，该枚举有 3 个值：Text 把 ComandText 属性的值理解为普通的 SQL 语句，以文本的方式执行；StoredProcedure 把 ComandText 属性的值理解为存储过程的名称，将以存储过程的方式执行；TableDirect 把 ComandText 属性的值解释为表的名称，即直接操作指定的表。
- Connection：获取或设置 SqlCommand 对象使用的数据库连接对象。
- Parameters：设置 SqlCommand 对象要执行的命令文本的参数列表，默认为一个空集合。

还有其他很多，这里不做介绍了。

当然不是用它来设置参数的，而是用它来干活的。这里也提供了一些方法供我们使用。

- ExecuteNonQuery()：执行一个无返回值的命令。这里注意了，命令无返回值可不是说这个方法无返回值。这个方法实际的返回值是 int 类型的值，返回的是执行命令的响应行数。
- ExecuteReader()：执行返回一个列表的命令，比如查询语句。这个方法的返回值是 SqlDataReader。SqlDataReader 是一个读取器，可以逐条读取查询结果。
- ExecuteScalar()：返回结果集中第一行的第一列。这里返回的是一个 object 类型，使用的时候自行转换类型。这个方法多用于做数据统计。

这里仅列出 3 个最常用的方法，其他的方法读者有兴趣可自己查资料学习。

本节我们就使用第一个方法来完成这个实例。

5.2.2 实例描述

酒店方面负责系统开发的经理告诉我们，在这个酒店管理系统中，要有会员管理这一功能。

其实会员对于很多商业系统都是很重要的一个功能。对于每个营销型的公司，客户就是命脉，而会员，就是公司最看重，也最具开发潜力的客户资源。

当然，会员信息收集也就显得非常重要。本实例就以酒店管理系统中的会员添加功能来讲解一下 SqlCommand 对象的使用。

酒店会员有以下基本属性：编号、姓名、证件类型、证件号码、性别、生日、籍贯、工作单位。

对应的数据库表结构如表 5-1 所示。

表 5-1　会员信息表Customer

列　名	类　型	是否为空	说　明
Id	int	否	主键，自增列
Number	nvarchar(10)	否	会员编号，唯一
FullName	nvarchar(30)	是	会员姓名全名
CertificateType	nvarchar(10)	是	证件类型
CertificateNumber	nvarchar(20)	是	证件号码
Sex	bit	是	性别
Birthday	datetime	是	生日
NativePlace	nvarchar(20)	是	籍贯
WorkUnit	nvarchar(60)	是	工作单位

5.2.3　实例应用

【例 5-6】添加酒店会员。

(1)　新建一个 Web 项目。

(2)　添加一个新的 Web 页面，命名为 AddCustomer.aspx。

(3)　修改页面视图，添加如下代码：

```
<table cellspacing="1" cellpadding="3">
<thead><tr><td colspan="2" align="center">添加会员</td></tr></thead>
<tbody>
<tr>
<td class="td_left">会员编号：</td>
<td>
<asp:TextBox ID="txtNumber" runat="server" MaxLength="10">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="td_left">会员姓名：</td>
<td>
<asp:TextBox ID="txtFullName" runat="server" MaxLength="30">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="td left">证件类型：</td>
<td>
    <asp:DropDownList ID="ddlCertificateType" runat="server">
        <asp:ListItem>身份证</asp:ListItem>
        <asp:ListItem>军官证</asp:ListItem>
        <asp:ListItem>护照</asp:ListItem>
```

```
    </asp:DropDownList>
</td>
</tr>
<tr>
<td class="td_left">证件号码：</td>
<td>
<asp:TextBox ID="txtCertificateNumber" runat="server" MaxLength="20">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="td_left">性别：</td>
<td>
    <asp:DropDownList ID="ddlSex" runat="server">
        <asp:ListItem Value="1">男</asp:ListItem>
        <asp:ListItem Value="0">女</asp:ListItem>
    </asp:DropDownList>
</td>
</tr>
<tr>
<td class="td_left">生日：</td>
<td>
<asp:TextBox ID="txtBirthday" runat="server">
</asp:TextBox>格式：yyyy-mm-dd 如：2001-12-1
</td>
</tr>
<tr>
<td class="td_left">籍贯：</td>
<td>
<asp:TextBox ID="txtNativePlace" runat="server" MaxLength="20">
</asp:TextBox>
</td>
</tr>
<tr>
<td class="td_left">工作单位：</td>
<td>
<asp:TextBox ID="txtWorkUnit" runat="server" MaxLength="60" Width="200">
</asp:TextBox>
</td>
</tr>
<tr>
<td colspan="2" align="center">
<asp:Button ID="btnSave" runat="server" Text=" 保存 "
  onclick="btnSave_Click" /><br />
<asp:Label ID="lblResultMessage" runat="server" ForeColor="Red"></asp:Label>
</td>
</tr>
</tbody>
</table>
```

(4) 给服务器控件 btnSave 添加 onclick 事件处理程序 btnSave_Click。代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    SqlConnection conn = null;    //创建连接对象
    try
    {
        //初始化连接字符串
        string connStr =
          "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";
        //初始化连接对象
        conn = new SqlConnection(connStr);

        //使用连接对象的 CreateCommand()方法创建 SqlCommand 对象
        //这里还可以 new 一个 SqlCommand 对象，设置它的 Connection 属性
        SqlCommand cmd = conn.CreateCommand();

        //初始化 SQL 语句
        StringBuilder sqlBuilder =
          new StringBuilder("insert into Customer values('");
        sqlBuilder.Append(this.txtNumber.Text).Append("','");    //编号
        sqlBuilder.Append(this.txtFullName.Text).Append("','");    //姓名
        sqlBuilder.Append(this
          .ddlCertificateType.Text).Append("','");    //证件类型
        sqlBuilder.Append(this
          .txtCertificateNumber.Text).Append("',");    //证件号码
        sqlBuilder.Append(this.ddlSex.Text).Append(",'");    //性别
        sqlBuilder.Append(this.txtBirthday.Text).Append("','");    //生日
        sqlBuilder.Append(this.txtNativePlace.Text).Append("','");    //籍贯
        sqlBuilder.Append(this.txtWorkUnit.Text).Append("')");    //工作单位
        string sql = sqlBuilder.ToString();    //生成 SQL 语句

        cmd.CommandText = sql;    //设置命令文本
        cmd.CommandType = CommandType.Text;    //设置命令类型

        conn.Open();    //在对数据库执行操作以前打开数据库
        int result = cmd.ExecuteNonQuery();
        conn.Close();    //执行完对数据库操作及时关闭数据库
        this.lblResultMessage.Text = "保存成功。数据库响应行数为:" + result;
    }
    catch (Exception ex)
    {
        //如果数据库连接没有关闭，关闭它
        if (conn.State == ConnectionState.Open) conn.Close();
        this.lblResultMessage.Text = ex.Message;    //在页面上显示异常信息
    }
}
```

这样添加会员功能就完成了，我们来运行一下。

5.2.4 运行结果

按键盘上的 F5 键，或者单击工具栏中的“启动调试”按钮，开始运行项目。

打开“添加会员”页面，接下来我们输入一组会员信息。单击“保存”按钮。页面提示“保存成功。数据库响应行数为:1”。页面效果如图 5-7 所示。



图 5-7 添加会员

这就说明我们的方法执行成功了。

还不太自信吗？来看一下数据库吧。

启动 SQL Server Management Studio，打开 hotel 数据库下面的 Customer 表，可以看到刚才我们保存的那条记录已经成功插入 Customer 表，如图 5-8 所示。

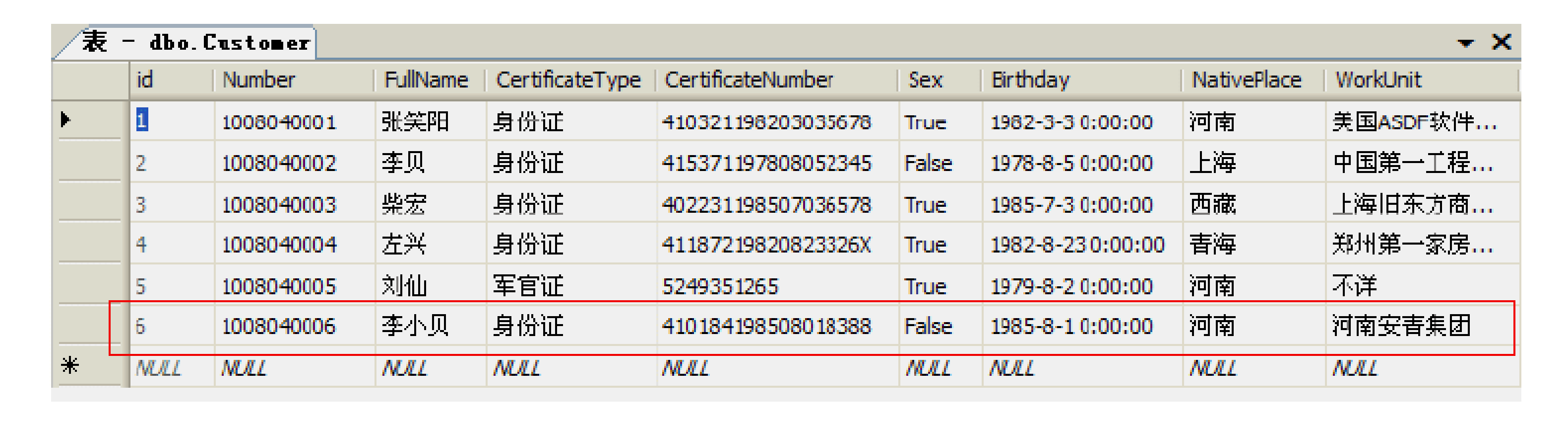


表 - dbo.Customer

id	Number	FullName	CertificateType	CertificateNumber	Sex	Birthday	NativePlace	WorkUnit
1	1008040001	张笑阳	身份证	410321198203035678	True	1982-3-3 0:00:00	河南	美国ASDF软件...
2	1008040002	李贝	身份证	415371197808052345	False	1978-8-5 0:00:00	上海	中国第一工程...
3	1008040003	柴宏	身份证	402231198507036578	True	1985-7-3 0:00:00	西藏	上海旧东方商...
4	1008040004	左兴	身份证	41187219820823326X	True	1982-8-23 0:00:00	青海	郑州第一家房...
5	1008040005	刘仙	军官证	5249351265	True	1979-8-2 0:00:00	河南	不详
6	1008040006	李小贝	身份证	410184198508018388	False	1985-8-1 0:00:00	河南	河南安青集团
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

图 5-8 会员信息已经成功插入数据库

5.2.5 实例分析



源码解析：

该实例先创建了一个数据库连接对象，然后使用该连接对象创建一个 SqlCommand 对象。接着我们使用 StringBuilder 对象和页面上的值拼接出一个 SQL 语句。接着初始化 SqlCommand 对象的 CommandText 和 CommandType 属性。然后打开连接，执行命令并返回执行结果，然后关闭连接，显示执行结果。

在整个执行过程可能会出现异常，所以这里需要进行异常捕获并进行处理。

5.3　列表显示数据库酒店会员信息

使用数据库存储数据，归根结底还是要使用它。要是只存不取，存储数据就没有意义了。不可能有哪个人闲得无聊写个系统胡乱存数据玩吧。

上一节我们讲了 SqlCommand 对象，学习了使用它怎么向数据库中存储数据。但它的功能肯定不只如此。

前面我们也提到过，SqlCommand 是一个“无微不至的助理”，既然都“无微不至”了，肯定不仅仅存储一下数据这么简单，它肯定还能干很多其他事情。

现在我们再来研究一下它怎样从数据库中取数据。

当然，我们从数据库取的数据往往不是一两个数字那么简单。所以我们的助理还要找其他工具来完成我们读取数据的操作，这就是我们要讲的 SqlDataReader。

 **视频教学：** 光盘/videos/5/SqlDataReader.avi　　　　长度：7 分钟

5.3.1　基础知识——SqlDataReader

SqlDataReader 就像一列火车。

当助理 SqlCommand 接到一个 ExecuteReader 的命令时，就知道老板要取很多货物回去了。它就随手叫来一列上面所说的那种火车 SqlDataReader，装着从数据库取的数据回来了。

回来以后站在公司门口，找人一节节车箱地把数据搬下来。搬完了，付车费，把火车打发走，取数据的操作就完成了。

但是，取数据的同时，运货物的桥 SqlConnection 千万不能拆了，你一拆了，火车一看你断它后路，让它回不去了，它就罢工了。

不过在你把火车打发走以后还是要记得拆桥的，也就是要记得关 SqlConnection。

SqlDataReader 类提供了一些方法供用户使用。常用的方法如下。

- Close()：关闭对象，释放资源。按上面例子中的说法也就是要把这列火车打发走。
- Read()：返回一个布尔型值，使 SqlDataReader 指针前进到下一条记录，如果前进成功，返回 true，否则返回 false。

还有一批以“Get+数据类型名”命名的方法，参数是整型数值。功能是从 Reader 里取出相对应位置的相应类型值。

上面提到了一种从 SqlDataReader 方法里取值的方法，不过遗憾的是它只能按位置取值，这样如果数据库列顺序改变的话对程序影响较大，我们可以根据情况选择使用。

下面我们来介绍另一种方式——SqlDataReader 索引器。这种方式可以有两种方法，根据表字段，或者和上面所说的方法一样根据字段索引。但是，这种方法也有一点是不很好的，即它的返回值是 object 类型的，我们还要进行类型转换工作。

不过我们倒觉得数据库字段位置变化的可能性要比字段类型或字段名变化的可能性大，所以会更喜欢根据表字段索引器方式取值。

我们来看一下根据表字段索引器取值的例子：

```
string fullName =
  reader["FullName"].ToString();    //这里的 reader 是 SqlDataReader 对象
```

下面来看一个实例。

5.3.2 实例描述

回到酒店管理系统。我们能向数据库添加会员信息了，当然也需要知道酒店现在所有的会员信息了。

这里就用到了数据查询功能。

刚才我们也说过了用 SqlDataReader 对象能从数据库取出大量数据信息，现在就试试用它来读取当前会员列表。

5.3.3 实例应用

【例 5-7】列表显示数据库酒店会员信息。

(1) 还是打开前面我们创建的 Web 项目。

(2) 新建一个页面，专门用于显示会员列表，取名为 ListCustomer.aspx。

(3) 修改页面视图，添加如下代码：

```
<table cellspacing="1" cellpadding="3">
<thead><tr><td  align="center">会员列表</td></tr></thead>
<tbody>
<tr><td>
   <asp:GridView ID="gvCustomer" runat="server"></asp:GridView>
   <!-- GridView 控件是一个数据列表控件，功能强大，这里暂不介绍，先这样用 -->
</td></tr>
</tbody>
</table>
```

(4) 这里需要对数据进行封装，所以用到一个实体类 CustomerInfo。代码如下：

```
public class CustomerInfo
{
   public CustomerInfo(){}    //构造方法

   public int ID { get; set; }    //ID
   public string Number { get; set; }    //编号
   public string FullName { get; set; }    //全名
   public string CertificateType { get; set; }    //证件类型
   public string CertificateNumber { get; set; }    //证件号码
   public bool Sex { get; set; }    //性别
   public DateTime Birthday { get; set; }    //生日
   public string NativePlace { get; set; }    //籍贯
```

```
    public string WorkUnit { get; set; }    //工作单位
}
```

(5) 打开页面后台的 cs 文件，修改 Page_Load 方法。代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //初始化连接字符串
    string connStr =
      "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";

    //初始化 SQL 语句
    string sql = "select * from Customer";
    //初始化一个集合，用于封装数据库取出的数据
    IList<CustomerInfo> customers = new List<CustomerInfo>();

    //使用 using 语句创建对象，使用完对象以后自动释放对象资源，不必显式释放了
    using (SqlConnection conn = new SqlConnection(connStr))
    {
        //使用构造方法创建 SqlCommand 对象，同时初始化 CommandText 和 Connection 属性
        SqlCommand cmd = new SqlCommand(sql, conn);
        conn.Open();    //打开数据库

        //执行 SqlCommand 对象的 ExecuteReader 方法获取 SqlDataReader 对象
        SqlDataReader reader = cmd.ExecuteReader();
        while (reader.Read())   //读一条记录
        {
            customers.Add(new CustomerInfo() {
                Birthday = DateTime.Parse(reader["Birthday"].ToString()),
                CertificateNumber = reader["CertificateNumber"].ToString(),
                CertificateType = reader["CertificateType"].ToString(),
                FullName = reader["FullName"].ToString(),
                ID = int.Parse(reader["ID"].ToString()),
                NativePlace = reader["NativePlace"].ToString(),
                Number = reader["Number"].ToString(),
                Sex = bool.Parse(reader["Sex"].ToString()),
                WorkUnit = reader["WorkUnit"].ToString()
            });
        }
        reader.Close();  //释放 SqlDataReader 资源

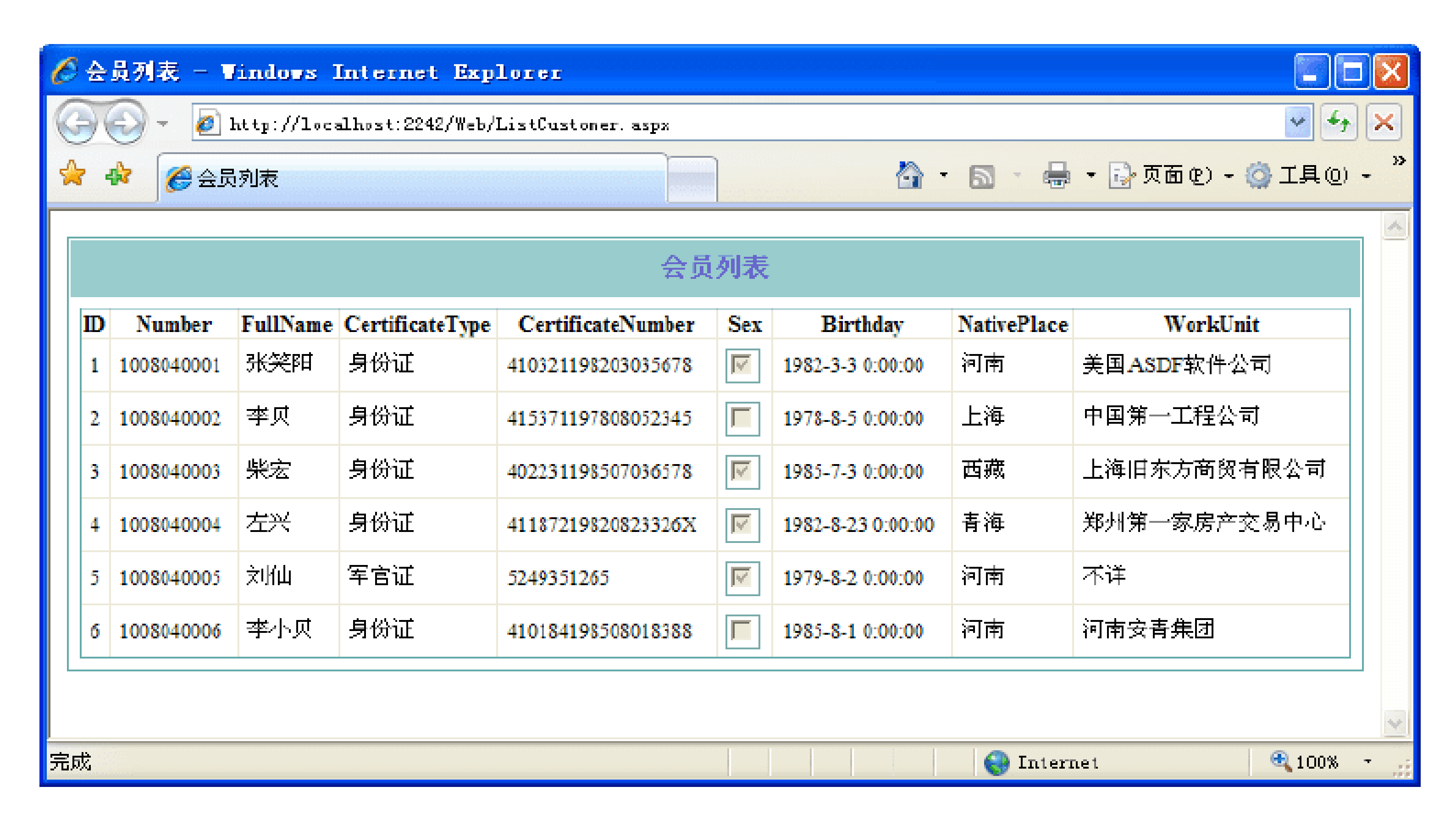
        this.gvCustomer.DataSource = customers;   //设置页面 GridView 控件的数据源
        this.gvCustomer.DataBind();    //重新绑定数据源
    }
}
```

这样就完了，不过注意一下。这里用到一个视图控件 GridView。它功能强大，要用好却很复杂，这里先不讲它，只照样子使用就好了。

接下来，我们运行该页面。

5.3.4　运行结果

编译、运行项目，访问 ListCustomer.aspx 页面。可以看到页面列出了所有的会员信息，如图 5-9 所示。



会员列表

ID	Number	FullName	CertificateType	CertificateNumber	Sex	Birthday	NativePlace	WorkUnit
1	1008040001	张笑阳	身份证	410321198203035678	☑	1982-3-3 0:00:00	河南	美国ASDF软件公司
2	1008040002	李贝	身份证	415371197808052345	☐	1978-8-5 0:00:00	上海	中国第一工程公司
3	1008040003	柴宏	身份证	402231198507036578	☑	1985-7-3 0:00:00	西藏	上海旧东方商贸有限公司
4	1008040004	左兴	身份证	41187219820823326X	☑	1982-8-23 0:00:00	青海	郑州第一家房产交易中心
5	1008040005	刘仙	军官证	5249351265	☑	1979-8-2 0:00:00	河南	不详
6	1008040006	李小贝	身份证	410184198508018388	☐	1985-8-1 0:00:00	河南	河南安青集团

图 5-9　会员列表

看看效果不错吧，对比图 5-8 中的数据库信息，它也都一条不少地列出来了。

5.3.5　实例分析



源码解析：

实例先初始化连接字符串、SQL 语句和保存结果的数据列表 customers。然后用连接字符串创建一个 SqlConnection 对象，再使用 SqlConnection 对象和 SQL 语句创建一个 SqlCommand 对象。接着打开数据库连接并执行 SqlCommand 对象的 ExecuteReader 方法获取 SqlDataReader。然后遍历该 SqlDataReader，取出并封装数据到列表 customers，然后关闭连接并绑定页面数据控件的数据源。

5.4　修改会员信息

上面两个实例我们已经完成对会员信息的最基本操作：添加与查询功能。不过百密一疏，任何人都可能一时手误输错一些信息。

比如万一酒店前台小姐把一个小伙子的性别选成“女”了。有一天酒店做优惠活动，送人家大小伙子一个女性用品，那还不气毁人家。不但说东西对别人有没有用处，还会影响酒店的声誉啊，这么不细心。

即使酒店前台小姐都很细心，但顾客可不一样。保不准哪天哪个顾客换工作了，或哪个证

件作废了。遇到这种情况，系统尽量还是要允许修改一些信息的。

所以，会员管理必须要有修改会员信息的功能。

刚才在会员添加的时候我们写的 SQL 字符串很长、很乱、很容易出错，使用这种方法确实相当痛苦。微软也意识到了这种痛苦，给出了一个很好的解决方案：SqlParameter。

所以这一节我们使用 SqlCommand 和 SqlParameter，以参数方式来设置我们的修改会员信息的 SQL 语句。



视频教学：光盘/videos/5/SqlParameter.avi　　　　长度：8 分钟

5.4.1　基础知识——SqlParameter

会员信息的修改功能比较简单，其实基本上和添加一样，无非是这里执行一个修改的 SQL 语句而已。

不过话又说回来了，回头看看我们前面写的代码，只有用一个词来形容：乱七八糟。

功能是实现了，不过所有代码写到一块了，回头再读起来确实特别费劲。总感觉好几件事放一块干了，没有一点条理性。这次我们就注意一点儿。

特别要强调的是那个 SQL 语句，一点点拼凑，一不小心就拼错了。那个还不很长，要是遇到复杂的语句，保不定哪里多个单引号，哪里又少个逗号呢。

突然想到 ADO.NET 好像给 Command 对象提供了一个参数机制，能用参数对象里的值自动替换 SQL 语句里的参数名，免去了一点点拼凑字符串的辛苦，会让 SQL 语句写得更加方便一点。最起码不怕格式写错了。基本上能解决 SQL 语句拼得不好看的问题，先用它试试吧。

找了找，果然在 SqlCommand 类中有一个属性 Paramenters，它是一个空的属性集合，存储的是 SqlParameter 对象，我们需要将自己创建的参数对象(SqlParameter 类的实例)全部添加到这个集合中就行了。

SqlParameter 提供了 7 个构造方法，我们来看其中最常用的两个。

- SqlParameter()：默认的构造方法，无参。
- SqlParameter(string parameterName, object value)：有两个参数。一个是参数名，就是在 SQL 语句里要替换掉的名称，一个是任意数据库支持的类型的对象。具体的类型由系统自动判断。

当然，还有其他一些更强大的构造方法。这里就不一一说明了，建议读者自己研究一下。

SqlParameter 类还提供了一些属性，供用户对它进行更详细的设置，我们来看一下最基本的两个。

- ParameterName：这个是参数名称，对应 SQL 语句里要替换掉的名称，默认为空字符串。
- Value：是 object 类型的对象，对应构造方法里的值，默认为 null。



这里的 ParameterName 参数的格式必须以@符号开头，当然，在 SQL 语句里也需要用同样的名称标记要替换的参数。

下面我们来演示一下怎样创建一个参数对象，并添加到 SqlCommand 对象的参数集合中。

假设有以下 SQL 语句：

```
string sql = "insert into users values(@UserName,@Password,@Age)";
```

我们就可以这样添加参数：

```
cmd.Parameters.Add(
  new SqlParameter("@UserName", "zhangsanfeng"));    //cmd 是 SqlCommand 对象
cmd.Parameters.Add(new SqlParameter("@Password", "3598541zsf"));
cmd.Parameters.Add(new SqlParameter("@UserName", 31));
```

顺便得赞叹一下 SqlParameter，好东西就是好东西。听说它还能够过滤 SQL 注入呢。

了解这些以后就可以完善我们的添加酒店会员功能了。

5.4.2 实例描述

很轻松地做了两个功能，看看时间也不早了，睡去吧。

嘴角一抹甜蜜的微笑，梦到我们的项目开发大大地提前了，提早进入了实施阶段。为此整个项目团队小小庆贺了一番。

在实施阶段我负责现场技术支持工作。

有一天来了一位顾客，是个转业军人，一看就很霸气的那种。拿了一个转业证明和新办的居民身份证要求更改会员的证件号码。因为听说酒店有优惠活动，会员集够 10000 分送一辆原装进口的悍马越野车。这位哥已经 9999 分了，却赶上转业了，想把会员信息的证件改成新办的身份证。

无奈酒店前台小姑娘却怎么也找不着修改功能，叫作者过去了。作者一听，也晕了。

“还没开发啊！”

刚一说，就看那位霸气的军人大哥把椅子抡过来了……

吓醒了，一身冷汗……

赶紧添上修改功能吧。

5.4.3 实例应用

【例 5-8】修改会员信息。

为了使代码功能更模块化，这个实例我们将使用 CustomerInfo 类封装会员信息。

(1) 首先打开上次我们使用的项目。

(2) 创建一个新的页面 EditCustomer.aspx。

(3) 页面视图代码就直接复制 AddCustomer.aspx 页面的了。

(4) 修改最好把原始的信息显示出来，让用户有选择性地改动。所以我们要在页面打开的时候向页面控件填充会员原始信息。

修改 Page_Load 方法如下：

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
    if (!IsPostBack)     //只有在第一次加载页面的时候执行
    {
        this.InitCustomer();     //初始化会员信息
    }
}
```

初始化工作比较复杂，我们把它封装起来。代码如下：

```
private void InitCustomer()
{
    /*
    * 初始化会员信息
    */

    //初始化连接字符串
    string connStr =
      "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";
    //初始化 SQL 语句，使用提交过来的 id 作为查询条件
    string sql = "select * from Customer where id=" + Request["id"];

    //使用 using 语句创建对象，使用完对象以后自动释放对象资源，不必显式释放了
    using (SqlConnection conn = new SqlConnection(connStr))
    {
        //使用构造方法创建 SqlCommand 对象，同时初始化 CommandText 和 Connection 属性
        SqlCommand cmd = new SqlCommand(sql, conn);
        conn.Open();     //打开数据库

        //执行 SqlCommand 对象的 ExecuteReader 方法获取 SqlDataReader 对象，
        //不必显式释放
        using (SqlDataReader reader = cmd.ExecuteReader())
        {
            if (reader.Read())    //读一条记录
            {
                //取出并封装会员信息
                CustomerInfo customer = new CustomerInfo()
                {
                    ID = int.Parse(reader["id"].ToString()),
                    Birthday = DateTime.Parse(reader["Birthday"].ToString()),
                    CertificateNumber = reader["CertificateNumber"].ToString(),
                    CertificateType = reader["CertificateType"].ToString(),
                    FullName = reader["FullName"].ToString(),
                    NativePlace = reader["NativePlace"].ToString(),
                    Number = reader["Number"].ToString(),
                    Sex = bool.Parse(reader["Sex"].ToString()),
                    WorkUnit = reader["WorkUnit"].ToString()
                };
                this.FullPage(customer);     //填充到页面
            }
            else
            {
```

```
                //如果读取失败，返回列表页面
                Response.Redirect("ListCustomer.aspx");
            }
        }
    }
}
```

页面填充功能比较独立，我们也封装起来：

```
private void FullPage(CustomerInfo customer)
{
    /*
    * 填充页面信息
    */
    this.ddlCertificateType.SelectedValue =
      customer.CertificateType;    //证件类型
    this.ddlSex.SelectedValue = customer.Sex ? "1" : "0";    //性别
    this.txtBirthday.Text =
      string.Format("{0:yyyy-MM-dd}", customer.Birthday);    //生日
    this.txtCertificateNumber.Text = customer.CertificateNumber;   //证件号码
    this.txtFullName.Text = customer.FullName;    //全名
    this.txtNativePlace.Text = customer.NativePlace;    //籍贯
    this.txtNumber.Text = customer.Number;    //会员编号
    this.txtWorkUnit.Text = customer.WorkUnit;    //工作单位
    ViewState["ID"] = customer.ID;    //编号
}
```

(5) 接下来开始实现修改功能，不过和添加也差不多。这次我们先封装一下会员信息：

```
private CustomerInfo GetCustomerForPage()
{
    /*
     * 封装页面的会员信息
     */
    return new CustomerInfo()
    {
        ID = int.Parse(ViewState["ID"].ToString()),    //ID
        Number = this.txtNumber.Text,    //编号
        FullName = this.txtFullName.Text,    //姓名
        CertificateType = this.ddlCertificateType.Text,    //证件类型
        CertificateNumber = this.txtCertificateNumber.Text,    //证件号码
        Sex = this.ddlSex.Text == "1",    //性别
        Birthday = DateTime.Parse(this.txtBirthday.Text),    //生日
        NativePlace = this.txtNativePlace.Text,    //籍贯
        WorkUnit = this.txtWorkUnit.Text    //工作单位
    };
}
```

仔细体会一下，想必已经感觉出来封装的好处了吧。各个属性数据类型在这里就进行一遍过滤，省得到数据库又出错了。

有一次就是因为在凑 SQL 语句的时候少了日期型数据的引号，结果“2010-8-8”这样的日期系统先做减法再保存了，郁闷了好久才找到是这个原因。

(6)　刚看了添加参数的语句有点长，不太好看，这里写一个方法对它封装一下：

```
private void SetParameter(SqlCommand cmd, string parameterName, object value)
{
    /*
    * 设置参数
    */
    cmd.Parameters.Add(new SqlParameter(parameterName, value));
}
```

这样我们直接调用这个方法就可以了，代码肯定会简洁很多。

(7)　接下来封装一下保存会员信息的数据库操作代码，用户信息我们也用 SqlParameter 类封装后添加到 SqlCommand 对象的参数列表。代码如下：

```
private int UpdateCustomer(CustomerInfo customer)
{
    //初始化连接字符串
    string connStr =
      "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";

    //初始化 SQL 语句
    string sql = new StringBuilder("update Customer set")
      .Append(" Number = @Number,")
      .Append(" FullName = @FullName,")
      .Append(" CertificateType = @CertificateType,")
      .Append(" CertificateNumber = @CertificateNumber,")
      .Append(" Sex = @Sex,")
      .Append(" Birthday = @Birthday,")
      .Append(" NativePlace = @NativePlace,")
      .Append(" WorkUnit = @WorkUnit")
      .Append(" where ")
      .Append("id = @ID").ToString();

    //初始化连接对象，这种方式不用显式释放资源
    using (SqlConnection conn = new SqlConnection(connStr))
    {
        //创建 SqlCommand 对象，初始化命令语句和连接对象
        SqlCommand cmd = new SqlCommand(sql, conn);

        //添加参数列表
        this.SetParameter(cmd, "@Number", customer.Number);
        this.SetParameter(cmd, "@FullName", customer.FullName);
        this.SetParameter(cmd, "@CertificateType", customer.CertificateType);
        this.SetParameter(
          cmd, "@CertificateNumber", customer.CertificateNumber);

        this.SetParameter(cmd, "@Sex", customer.Sex);
```

```
        this.SetParameter(cmd, "@Birthday", customer.Birthday);
        this.SetParameter(cmd, "@NativePlace", customer.NativePlace);
        this.SetParameter(cmd, "@WorkUnit", customer.WorkUnit);
        this.SetParameter(cmd, "@ID", customer.ID);

        conn.Open();    //在对数据库执行操作以前打开数据库
        return cmd.ExecuteNonQuery();
    }
}
```

果然，看看干净利索的参数设置列表，耳目一新。一眼就能明白每一句我们在干什么，连注释都不用加了。

(8) 该封装的都封装完了，再来看一下事件处事程序的主方法。代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        CustomerInfo customer = this.GetCustomerForPage();
        int result = this.UpdateCustomer(customer);    //保存会员信息
        this.lblResultMessage.Text =
          "保存成功。数据库响应行数为:" + result;  //显示成功信息
    }
    catch (Exception ex)
    {
        this.lblResultMessage.Text = ex.Message;    //显示异常信息
    }
}
```

单击事件主过程简洁明了，看起来也不讨厌了。

好了，修改功能做完了，但总不能每次都在浏览器地址栏访问吧，我们在会员列表里添加一列“修改”，点击哪一行的“修改”链接，就修改那个用户。

(9) 打开 ListCustomer.aspx 页面，修改 GridView 控件的代码如下：

```
<asp:GridView ID="gvCustomer" runat="server">

    <Columns>
        <asp:TemplateField HeaderText="修改">
            <ItemTemplate>    <!-- 项模版 -->
                <!-- 一个超链接，绑定 ID 值生成链接路径 -->
                <a href='<%# "EditCustomer.aspx?id=" + Eval("ID") %>'>修改</a>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>

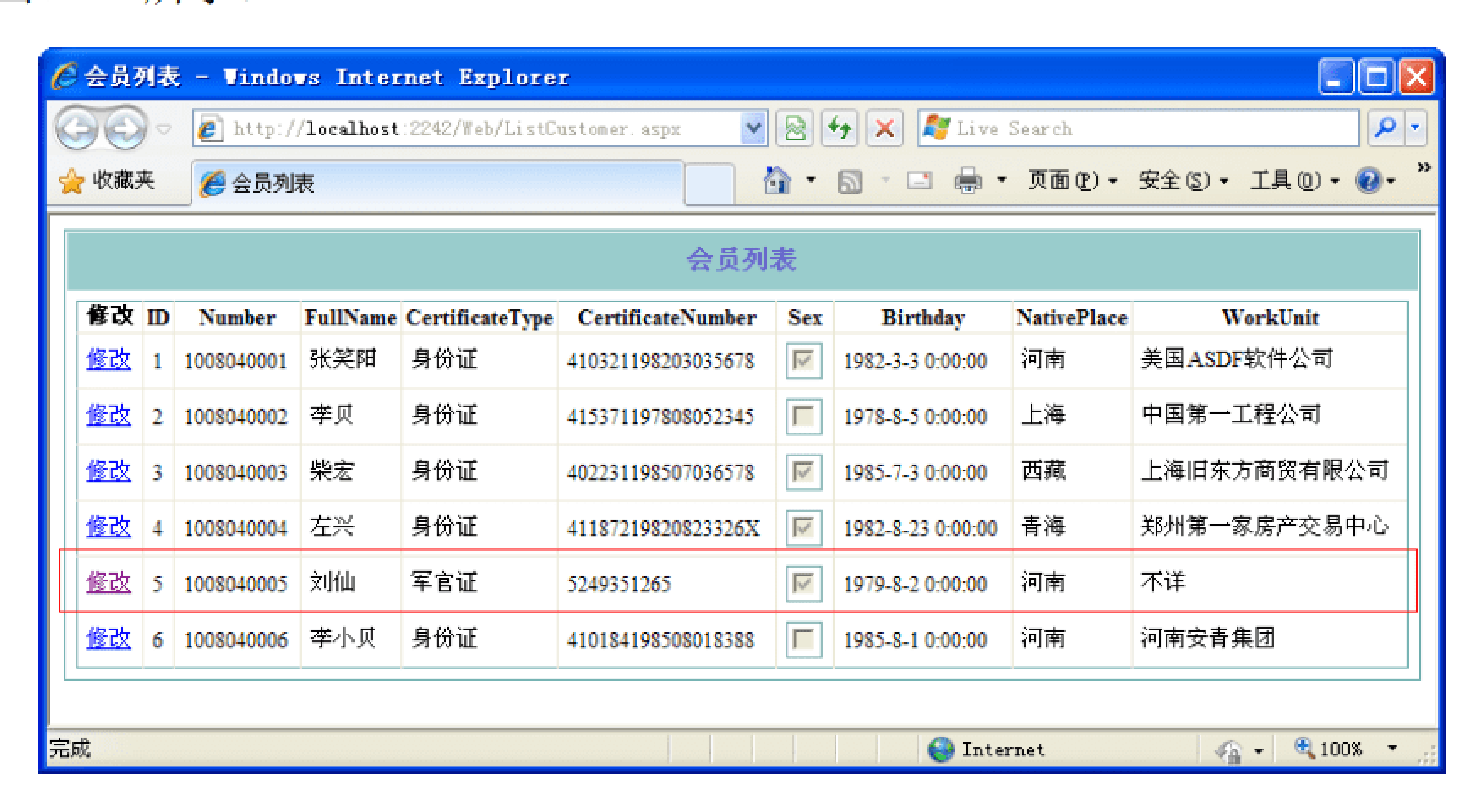
</asp:GridView>
```

服务器控件先不讲，这里就先这样写。正常情况下会有自动提示，也不用怕敲错键盘。

我们来运行一下看看。

5.4.4　运行结果

运行项目，访问 ListCustomer.aspx 页面。打开会员列表，我们看到了那位“霸气”的军官大哥，如图 5-10 所示。



修改	ID	Number	FullName	CertificateType	CertificateNumber	Sex	Birthday	NativePlace	WorkUnit
修改	1	1008040001	张笑阳	身份证	410321198203035678	☑	1982-3-3 0:00:00	河南	美国ASDF软件公司
修改	2	1008040002	李贝	身份证	415371197808052345	☐	1978-8-5 0:00:00	上海	中国第一工程公司
修改	3	1008040003	柴宏	身份证	402231198507036578	☑	1985-7-3 0:00:00	西藏	上海旧东方商贸有限公司
修改	4	1008040004	左兴	身份证	41187219820823326X	☑	1982-8-23 0:00:00	青海	郑州第一家房产交易中心
修改	5	1008040005	刘仙	军官证	5249351265	☑	1979-8-2 0:00:00	河南	不详
修改	6	1008040006	李小贝	身份证	410184198508018388	☐	1985-8-1 0:00:00	河南	河南安青集团

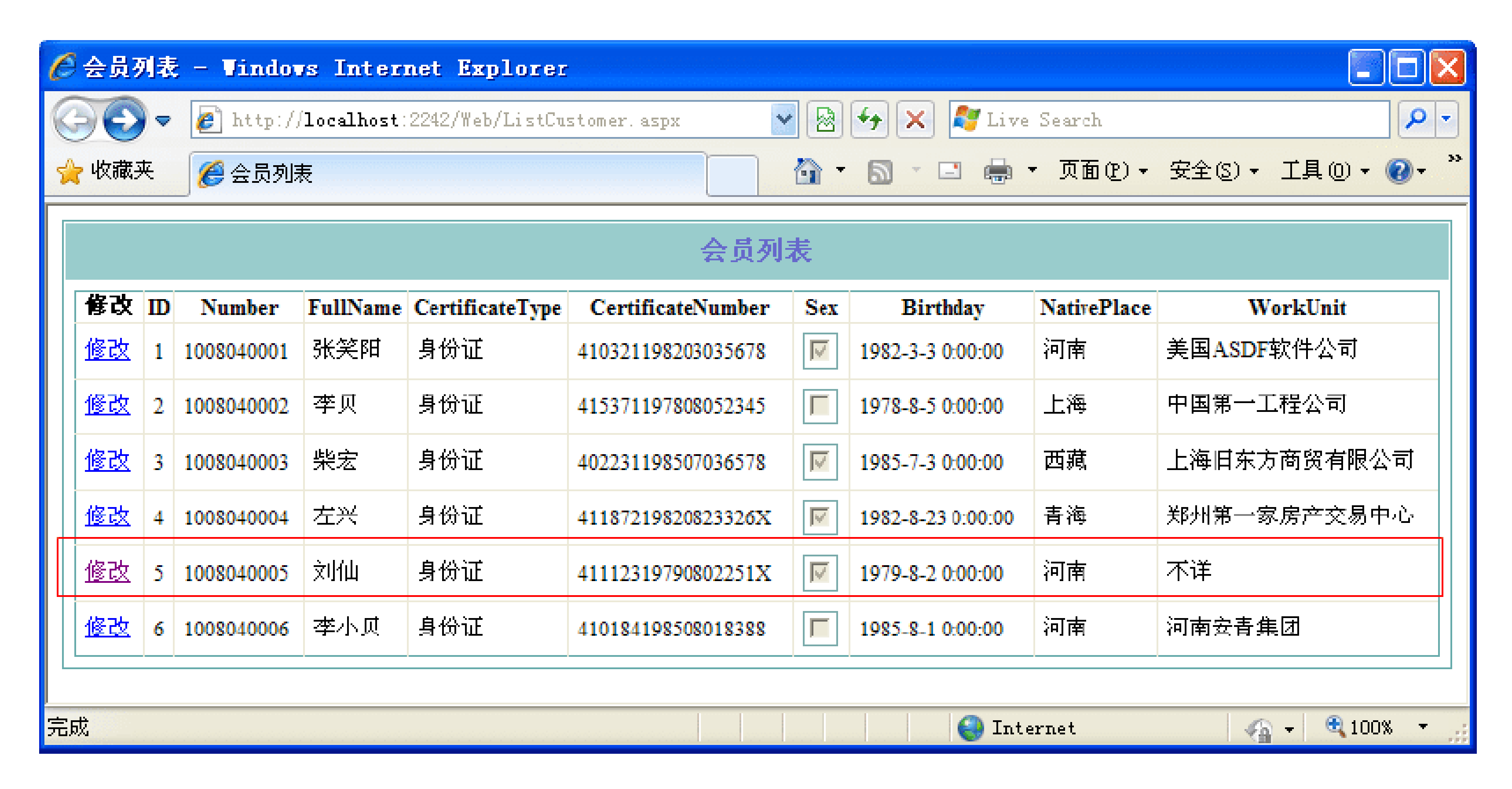
图 5-10　会员列表

单击这条记录第一列的“修改”超链接，来到会员编辑页面。我们修改“证件类型”为身份证，修改“证件号码”为相应的号码。单击“修改” 按钮。页面提示“保存成功。数据库响应行数为:1”。页面效果如图 5-11 所示。



图 5-11　修改会员信息

再回到会员列表页面，可以看到证件信息已经改过了，现在已经是我们修改的身份证了，如图 5-12 所示。



修改	ID	Number	FullName	CertificateType	CertificateNumber	Sex	Birthday	NativePlace	WorkUnit
修改	1	1008040001	张笑阳	身份证	410321198203035678	☑	1982-3-3 0:00:00	河南	美国ASDF软件公司
修改	2	1008040002	李贝	身份证	415371197808052345	☐	1978-8-5 0:00:00	上海	中国第一工程公司
修改	3	1008040003	柴宏	身份证	402231198507036578	☑	1985-7-3 0:00:00	西藏	上海旧东方商贸有限公司
修改	4	1008040004	左兴	身份证	41187219820823326X	☑	1982-8-23 0:00:00	青海	郑州第一家房产交易中心
修改	5	1008040005	刘仙	身份证	41112319790802251X	☑	1979-8-2 0:00:00	河南	不详
修改	6	1008040006	李小贝	身份证	410184198508018388	☐	1985-8-1 0:00:00	河南	河南安青集团

图 5-12　会员列表

5.4.5　实例分析



源码解析：

实例先接收提交过来的会员 id，根据 id 获得相应的值并填充到页面。

在修改会员信息的方法 UpdateCustomer 里首先初始化了一个连接字符串和带参数名称的 SQL 语句。然后创建 SqlConnection 对象和 SqlCommand 对象，接下来给 SqlCommand 对象设置参数，最后打开连接并执行命令返回结果。

5.5　列表查看房间信息

在酒店系统里，房间信息是最常用的一组信息了，我们经常要对它进行查询操作。

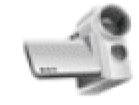
比如来了位顾客，想要一个“高级海景房”，我们需要查询一下所有“高级海景房”的入住信息。一看没空房了，顾客只好说换一个“豪华海景房”。一查，不错还有，就是有点贵，价格顾客接受不了了，也罢。换一个……

选了几次，没合适的，干脆那样，查询价格在 1000 以内的海景房。一查还不错，有两间普通海景房，环境还不错，带顾客看了房，定下来了……

像这样短时间内会有大量的数据库查询操作，对数据库压力可能会很大。

或许一般酒店管理系统业务量并不很大，所以对数据库访问的压力可以忽略。但这里假设我们的服务器要服务一个全球连锁，上千家分店，数据集中管理的一个酒店管理系统。那样就要考虑数据库服务器性能了。

所以像上面的一系列操作，我们可以把每个分店的房间信息保存到内存中，供以后再次访问查询。这里一个比较好的解决方案就是 DataSet。

 **视频教学：**光盘/videos/5/DataSet.avi　　长度：5 分钟

5.5.1　基础知识——DataSet和DataAdapter

DataSet，看名字就知道它是一个数据集合，但它有别于一般 Collection。它是一个存储在内存中的临时数据库，它有着基本上与数据库一样的结构，如图 5-13 所示。

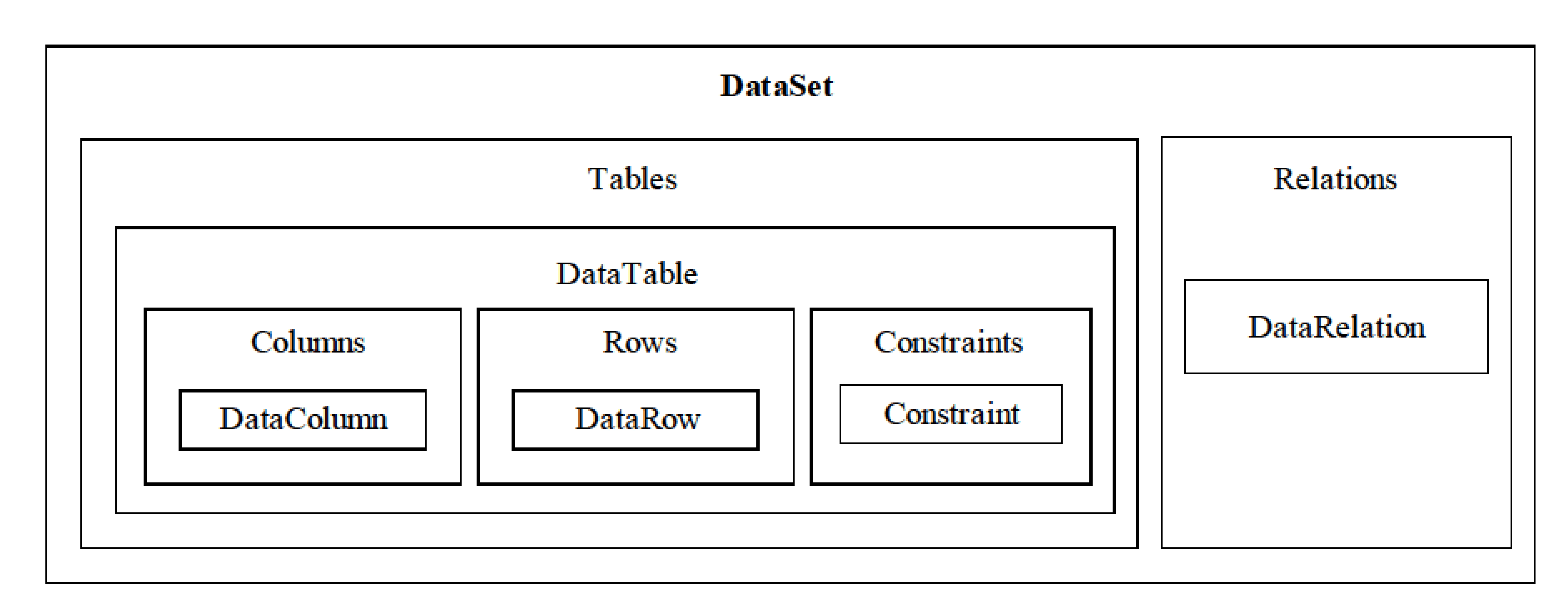


图 5-13　DataSet结构图

DataSet 包含一个 Tables 属性，它是对应于数据库表的 DataTable 对象的集合。

每个 DataTable 又可以包含对应记录行的 DataRow 对象集合 Rows 属性和对应表字段列的 DataColumn 对象集合的 Columns 属性，以及对应表约束的 Constraint 对象的集合 Constraints 属性。DataSet 还包含一些对应于 SQL 关系的 DataRelation 对象的集合 Relations 属性。

DataSet 其实内部是用 XML 来描述数据的，所以 DataSet 能用来存储不基于任何数据库的复杂关系型数据，而且不依赖数据库连接。

DataSet 一般有两种用法。

(1)　使用 DataAdapter 对象操作 DataSet 对象，实现对 DataSet 的数据填充，或用 DataSet 对象里的数据更新数据库。

(2)　把 XML 数据流或文本加载到 DataSet 对象。

其中最常用的是第一种，以数据库为数据来源的方法。

前面说了，DataSet 是一个临时数据库，为解决数据库和 DataSet 之间的数据同步问题，ADO.NET 针对各种数据库各提供了一个类来专门处理该问题。对于 SQLServer 数据库，就是 SqlDataAdapter 了。

SqlDataAdapter 是一个适配器，是在数据库和 DataSet 之间的一个桥梁，用以协调双方数据同步。它提供了 4 个构造方法如下。

- SqlDataAdapter()：初始化一个 SqlDataAdapter 类的实例。
- SqlDataAdapter(SqlCommand)：初始化一个 SqlDataAdapter 类的实例，这里用指定的 SqlCommand 作为 SelectCommand 的属性。
- SqlDataAdapter(String, SqlConnection)：使用 SelectCommand 和 SqlConnection 对象初始化一个 SqlDataAdapter 类的实例。
- SqlDataAdapter(String, String)：用 SelectCommand 和连接字符串初始化 SqlDataAdapter 类的实例。

以下是 SqlDataAdapter 类的常用属性。

- DeleteCommand：获取或设置一个 SQL 语句或存储过程，以从数据集删除记录。
- InsertCommand：获取或设置一个 SQL 语句或存储过程，以在数据源中插入新记录。
- SelectCommand：获取或设置一个 SQL 语句或存储过程，用于在数据源中选择记录。
- UpdateCommand：获取或设置一个 SQL 语句或存储过程，以更新数据源中的记录。
- TableMappings：获取一个集合，它提供源表和 DataTable 之间的主映射。

有两个最常用的方法。

- Update：为 DataSet 中每个已插入、已更新或已删除的行调用相应的 INSERT、UPDATE 或 DELETE 语句。
- Fill：填充 DataSet 或 DataTable。

这里我们使用得较为简单，将数据库的信息填充到 DataSet 后在页面上显示出来就可以了。代码如下：

```
//初始化 SqlDataAdapter。sql 是 SQL 语句，conn 是数据库连接对象
SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);
DataSet set = new DataSet();  //初始化 DataSet
adapter.Fill(set);  //填充数据
```

是不是看到了一个很稀奇的事情。它竟然不用打开数据库，也不用关闭，省了释放资源的操作了。

当然不是说它不用打开数据库就能取数据，其实在执行 Fill 方法时，系统会自动打开连接，使用完时自动关闭。打开和关闭连接的操作就省得用户自己写了。

5.5.2 实例描述

在酒店管理系统中，最最重要的，比会员还要重要的信息就是房间了。

房间是酒店的主营产品，是酒店的命根。同样，也是我们这个酒店管理系统中的核心元素，一切都是围绕它而存在的。

这里我们就先用 SqlDataAdapter 将酒店的房间信息填充到 DataSet 中，然后以 DataSet 为数据源将房间信息列表展示在 Web 页面上。

酒店房间有以下基本属性：编号、价格、分类、状态。

对应数据库表的结构如表 5-2 所示。

表 5-2 房间信息表Room

列　名	类　型	是否为空	说　明
Id	int	否	主键，自增列
Number	nvarchar(5)	否	房间编号，唯一
Price	int	是	房间价格
Class	nvarchar(20)	是	房间分类
Status	int	是	状态

5.5.3　实例应用

【例 5-9】列表查看房间信息。

(1)　首先打开以前我们使用的项目。

(2)　添加一个新的 Web 窗体，这里取名为 ListRoom.aspx。

(3)　在 ListRoom.aspx 的视图页面中拖入一个服务器端控件 GridView，并修改 ID 属性为 gvRoom。代码如下：

```
<asp:GridView ID="gvRoom" runat="server"></asp:GridView>
```

(4)　打开后台代码 ListRoom.aspx.cs 文件，添加一个绑定列表数据的方法 BindRoomList。代码如下：

```
private void BindRoomList()
{
    /*
     * 绑定房间列表
     */

    //初始化连接字符串
    string connStr =
      "Data Source=HZ;Initial Catalog=hotel;User ID=sa;Password=sa123";
    //初始化 SQL 语句
    string sql = "select * from Room";

    //使用连接字符串初始化数据库连接
    SqlConnection conn = new SqlConnection(connStr);
    //使用 SQL 语句和数据库连接对象初始化数据库连接
    SqlDataAdapter adapter = new SqlDataAdapter(sql, conn);
    //初始化一个备用存储数据的 DataSet 对象
    DataSet set = new DataSet();
    //填充 DataSet
    adapter.Fill(set);

    this.gvRoom.DataSource = set.Tables[0];  //设置数据源
    this.gvRoom.DataBind();  //绑定数据源
}
```

(5)　要实现页面在打开的时候绑定房间列表，所以要修改 Page_Load 方法，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        this.BindRoomList();  //绑定房间列表
    }
}
```

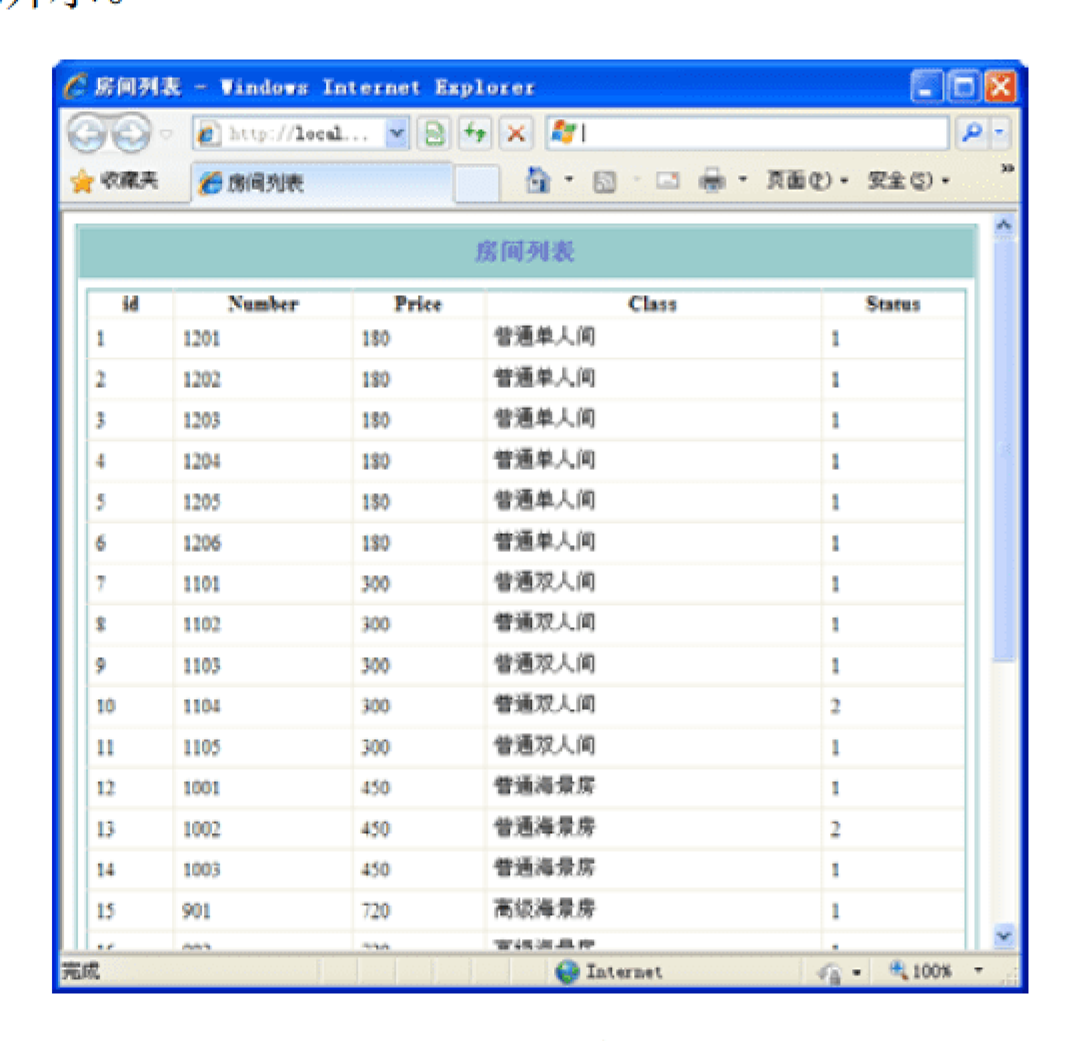
编码工作大功告成！下面我们来运行。

5.5.4　运行结果

生成网站，单击工具栏中的“启动调试”按钮。

访问 ListRoom.aspx 页面，正确执行。

运行结果如图 5-14 所示。



id	Number	Price	Class	Status
1	1201	180	普通单人间	1
2	1202	180	普通单人间	1
3	1203	180	普通单人间	1
4	1204	180	普通单人间	1
5	1205	180	普通单人间	1
6	1206	180	普通单人间	1
7	1101	300	普通双人间	1
8	1102	300	普通双人间	1
9	1103	300	普通双人间	1
10	1104	300	普通双人间	2
11	1105	300	普通双人间	1
12	1001	450	普通海景房	1
13	1002	450	普通海景房	2
14	1003	450	普通海景房	1
15	901	720	高级海景房	1

图 5-14　房间列表

5.5.5　实例分析



源码解析：

实例先声明了一个连接字符串和一个执行查询的 SQL 语句，然后创建并实始化一个数据库连接对象，然后使用 SQL 语句和数据库连接对象作为参数创建一个 SqlDataAdapter 对象。接着创建一个数据集 DataSet 对象，然后使用 SqlDataAdapter 对象的 Fill 方法填充数据集，最后将填充过的数据集绑定到页面数据控件。

5.6　使用ORM框架简化房间管理的数据访问操作

无论如何俭省，用 ADO.NET 访问数据库时都需要那些又臭又长的数据访问代码。

(1) 初始化连接字符串，创建连接。

(2) 创建一个命令对象或适配器对象。

(3) 得到一个读取器或者填充一个数据集。

(4) 把数据读取出来。

有需要的话，还得关闭对象、释放资源。

更可怕的是每次数据库操作我们都要写一遍。

清高一点地说，这些东西太没技术含量了。费时、费事、费心、费力。除了能锻炼我们敲键盘的速度外，实际上很难会得到什么好的经验或者其他好处。

当然，被这个问题困扰的并不是我们自己，一些先人大哲们早就为此绞尽了脑汁。

毕竟人家是“大哲”，对此问题早就给出了一些很好的解决方案。在 Java 中，Hibernate 已经取得了非凡的成就，而且那些伟大的先驱们也考虑到了我们.NET 程序员的痛苦，一并推出了 Hibernate 的.NET 版本——NHibernate。

不过因为它复杂的配置和因为其强大的功能而不得不做的一些独到的设计，使初学者入门稍微有些困难。

不过，微软一向是做“傻瓜”软件而处于不败之地的，这个问题当然也有相当好的解决方案——Linq to SQL。

Linq to SQL 也是一个 ORM 框架。

技术文档	ORM框架
ORM，全称为 Object/Relation Mapping，即对象/关系映射。通俗来讲就是实体对象到关系型数据库表的映射。 ORM 框架就是根据此思想编写出来的一套类库，可以很方便地实现对实体对象到关系型数据库之间数据增、删、改、查及数据封装的操作。它所做的工作就是把实体对象和数据库表对应起来，自动地对数据库进行存取操作。	

在 VS2008 以上的版本中，已经集成了对 Linq to SQL 的支持，我们可以直接使用它。

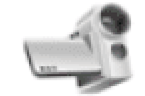
刚才提到了 NHibernate，同样也是一个 ORM 框架。因为我们总会习惯性地对同一类产品做一下比较，所以这里简单地发表一下作者的观点。

NHibernate 是一个优秀的框架，单看它的兄弟 Hibernate 在 Java 领域取得的巨大成功，我们就不可以小视它。Linq to SQL 出自微软，最大特点是使用简单方便，而这点刚好是前者所不具备的。

不过网上有不少前辈说前者要比后者功能强大，或许是的。不过即使强大，强大的地方肯定也是微乎其微的。不用举例，如果功能上差太远了，微软软件帝国的颜面何存啊？



因为 Linq to SQL 先天性不足，只支持 SQL Server 数据库(连微软自己的产品 Access 都不支持)，所以这里对 NHibernate 和 Linq to SQL 二者的比较都是基于 SQL Server 数据库而言的。



视频教学：光盘/videos/5/linq.avi　　　　长度：4 分钟

5.6.1　基础知识——Linq to SQL

因为作者以前学习 Java 的原因，所以是从 NHibernate 转过来的。

刚开始作者以为 Linq to SQL 一样也需要很多的配置。最起码，实体类和实体类与数据库

的对应关系总不可少吧。

事实上确实如此。不过，很可惜，VS2008 已经全部把它们封装了。

在 VS2008 以上的版本中，我们不用写一行代码(错了，我们根本不用写一个字符)，就可以完成对 Linq to SQL 的配置。

VS2008 提供了一个模板项来创建对 Linq to SQL 的配置，如图 5-15 所示。

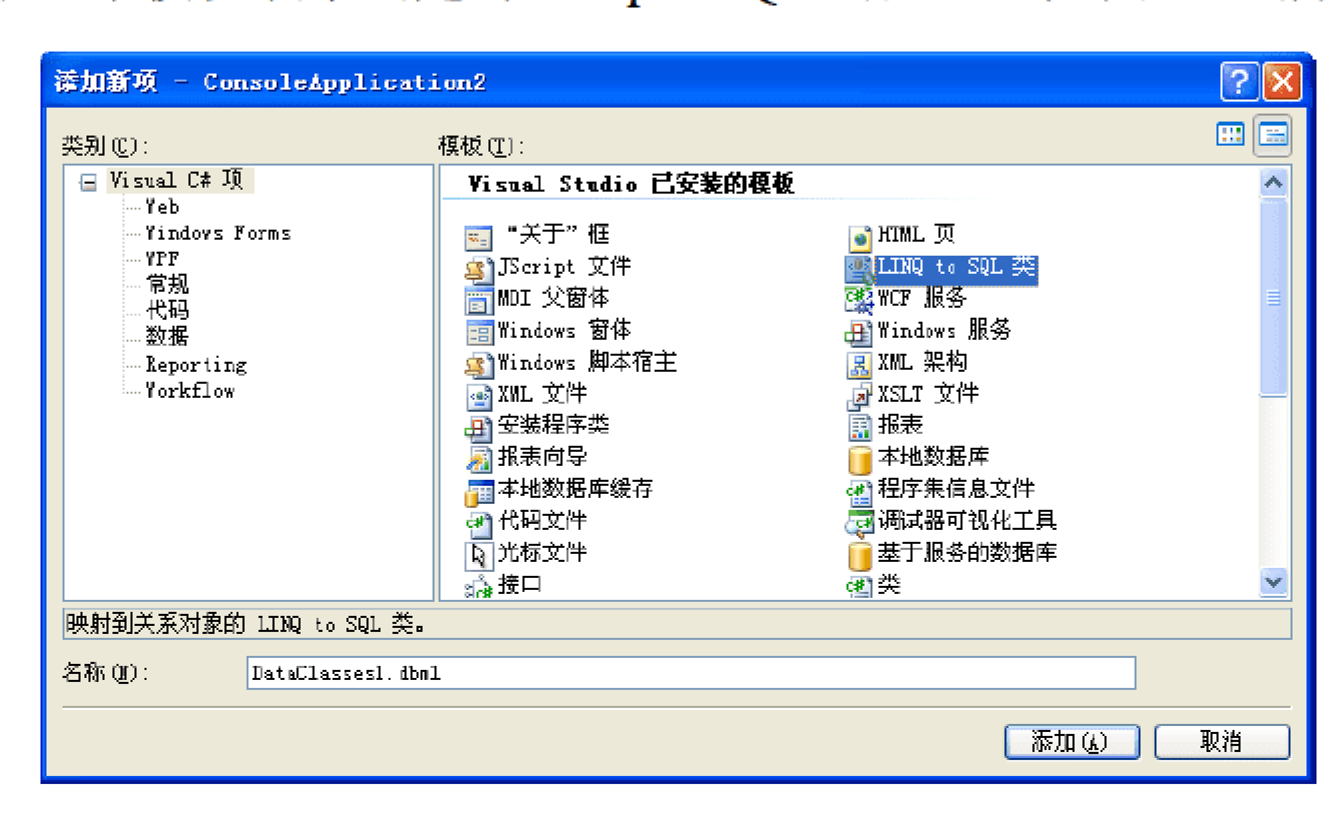


图 5-15 “添加新项”对话框

添加一个“Linq to SQL 类”以后，把“服务器资源管理器”里的“数据连接”下的对应数据库表拖放进来，就可以完成对这些数据库表的 ORM 配置，当然还有对应实体的创建。

使用它就非常简单了，可能简单到你不敢想象。

当然先引入刚才所创建的“Linq to SQL 类”所在的命名空间，还需要创建一个数据环境对象(DataClassesDataContext 类的实例)，代码如下：

```
private static DataClassesDataContext dataContext =
  new DataClassesDataContext();
```

好了，现在万事俱备，如何使用呢？

1. 查询

比如要得到酒店管理系统里所有的房间列表，可以这样：

```
dataContext.Room
```

确实就这样就行了，这就是所有数据库房间信息的列表。

这个属性是 System.Data.Linq.Table<Room>类型的一个集合，我们可以直接用 foreach 语句来遍历它，如下所示：

```
foreach (Room room in dataContext.Room)
{
    /* 这里是处理代码 */
}
```

当然，我们引入命名空间的时候已经把实体类引用过来了，这里可以直接使用了。

那只想取一条记录时怎么办呢？当然有好的办法，代码如下：

```
dataContext.Room.Single(r => r.Number == "1001");
```

上面这条语句查询出了房间号为 1001 的记录。顾名思义，这个方法只返回数据表中满足指定条件的唯一记录，如果满足条件的有多条记录存在，将会引发异常。

这里我们用到的 Single 方法的参数有点奇怪了，它是个 lambda 表达式。它就像一个简单的匿名函数，“=>”以前是函数参数，后面是函数逻辑。这里是遍历房间列表的时候把每个房间信息传递给参数 r，判断房间的 Number 属性值，如果等于 1001，就添加到 Single 方法的返回值集合。

单条的查询方法会了，我们想要查询符合条件的一个列表时怎么办呢？这就要用到 Linq 里比较先进的东西了。

Linq 允许我们像操作数据库一样来操作内存中的数据，来看代码：

```
var rooms = from r in dataContext.Room
          where r.Price > 300
          orderby r.id descending
          select r;
```

这条语句就可以取出所有入住价格在 300 元以上的房间信息，并根据房间的 ID 倒序排序。

熟悉 T-SQL 的读者面对该语句时应该一看就懂，基本上是与 T-SQL 一模一样的语法。不过这里要求必须以 select 或者是 group by 结束，所以 select 要写在后面。

这样一来，分页也很简单：

```
var rooms1 = rooms.Skip(3).Take(3).ToList();
```

一看就知道啥意思。房间集合跳过前 3 条，取 3 条，最后生成一个有三条信息的列表。

简单吧！

好了，查询基本上就是这样了，我们来看看添加。

2. 添加

代码如下：

```
//创建房间对象
Room newRoom = new Room()
{
    /* 属性设置略 */
};
//在提交更改的时候插入房间信息
dataContext.Room.InsertOnSubmit(newRoom);
//提交更改
dataContext.SubmitChanges();
```

得到一个实体对象以后，就两行代码，准备一个插入动作，执行上面所有准备的动作。

提交更改，该操作将提交对数据库的所有更改操作，包括添加、删除、修改。所以用 Linq to SQL 执行数据库的增、删、改操作时都需要执行提交更改操作。

3. 删除

删除操作也很简单，也只需要一句话，格式为：

```
dataContext.Room.DeleteOnSubmit(roomObject);    //roomObject 为房间类的实例
```

不过 DeleteOnSubmit 方法的参数是一个实体类，我们得先把它从数据库中取出来，而且执行完该方法并不能删掉，还要提交更改。

所以删除对象最终的代码为：

```
//取出房间实体
Room room = dataContext.Room.Single(c => c.Number = "1201");
//执行删除操作
dataContext.Room.DeleteOnSubmit(room);
//提交更改
dataContext.SubmitChanges();
```

也完成了，这段代码实现了删除房间号为 1201 的房间信息。

非常简单，除了注释就 3 行代码。

4. 修改

当然，修改时也要先取出来该条记录，修改完再保存。

代码如下：

```
//取出房间实体
Room room = dataContext.Room.Single(c => c.Number == "1201");
//修改房间状态
room.Status = 2;
//提交更改
dataContext.SubmitChanges();
```

这几行代码就能将 1201 号房间的状态编码改为 2 了。

5.6.2 实例描述

武侠小说里经常会有这样一幕：一个帅气的小伙子带着一位小巧可人的美女来到某市里最好的一家客栈。一进门就喊道："小二，开一间上好的厢房！"……

可见酒店最核心的业务是房间出租，所以在酒店管理系统里房间是必不可少的元素。

前面几节我们已经使用 ADO.NET 完成一部分功能，对数据库操作的过程写得真是痛不欲生。所以这一节我们就使用 Linq to SQL 代替 ADO.NET 实现对数据库操作的功能，来实现对房间信息的管理。

5.6.3 实例应用

【例 5-10】使用 ORM 框架简化房间管理的数据访问操作。

1) 构建 ORM 环境

(1) 打开上节我们使用的项目。

(2) 为了保持项目代码的整洁，我们将对 SQL 操作的类和配置信息独立到一个项目中来。这里创建一个类库项目，命名为 hotel.SQL。

(3) 在我们新建的项目 hotel.SQL 里添加一个“Linq to SQL 类”，命名为 DataClasses.dbml。

(4) 选择“视图”菜单下的“服务器资源管理器”命令，打开“服务器资源管理器”窗口，添加一个新的数据连接，连接到我们的酒店管理系统数据库。创建完以后在“服务器资源管理器”窗口里的“数据连接”节点下会多一个刚才我们创建的连接。

(5) 展开我们创建的连接下的“表”目录，会看到我们数据库所有的表。在这里把我们所有的表拖到打开着的 DataClasses.dbml 文件的设计区，即可完成对数据库的 ORM 映射。

这里暂时只有三个表，执行该操作后的 DataClasses.dbml 文件如图 5-16 所示。

上面完成了对数据库的 ORM 映射。

这时预示着我们完成了所有对数据库的操作代码。下面来实现我们的业务。

(6) 保存上面的操作，我们的 ORM 环境构建就完成了。

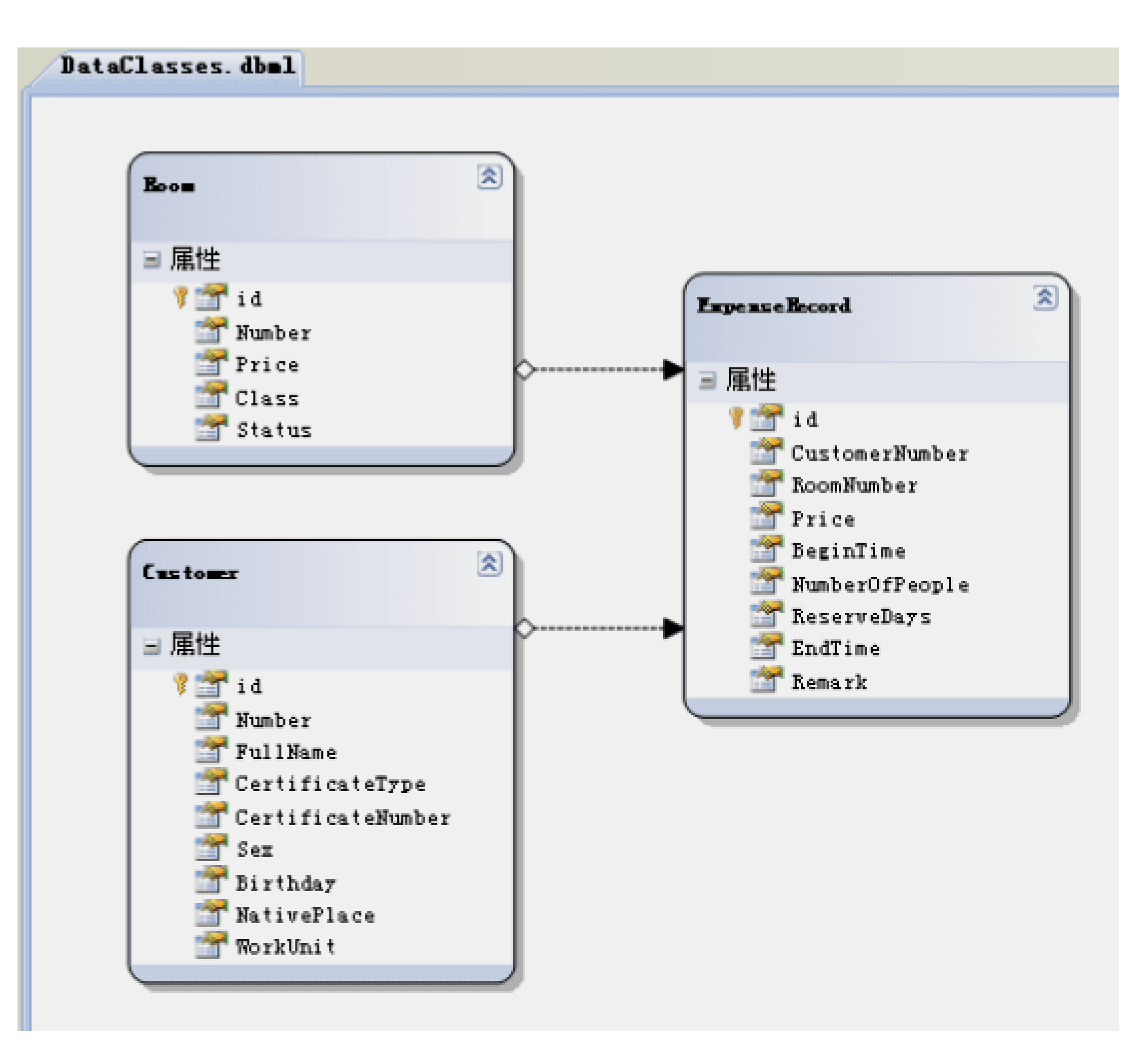


图 5-16　ORM映射

2) 使用 Linq to SQL 实现房间列表

(1) 回到我们的 Web 项目中，接下来要对房间列表页面的后台访问数据库代码进行重构，打开 ListRoom.aspx.cs 文件。

(2) 当然先要添加对 hotel.SQL 项目的引用，然后添加文件对命名 hotel.SQL 空间的引用。

(3) 修改绑定房间列表 BindRoomList 方法，代码如下：

```
private void BindRoomList()
{
    /*
    * 绑定房间列表
    */

    DataClassesDataContext dataContext =
      new DataClassesDataContext();    //初始化数据上下文

    //按价格排序
    var rooms = from r in dataContext.Room
```

```
                orderby r.Price
                select r;

    this.gvRoom.DataSource = rooms;  //设置数据源
    this.gvRoom.DataBind();  //绑定数据源
}
```

(4) 修改完后，保存，即可完成对房间列表的绑定。

可以看得出来，以前絮絮叨叨的一大段代码，在这里两行就搞定了。而且自始至终我们都没有去写一行操作数据库的代码。甚至它们已经把所有的数据都封装成对象给我们用了，而且对象用的实体类也都自动帮我们创建完毕。

3) 实现房间信息添加功能

(1) 先创建一个新的 Web 页面，命名为 AddRoom.aspx。

(2) 修改页面视图，添加两个文本框控件，一个用于输入编号，一个用于输入价格；再添加两个列表框控件，一个用于选择房间类型，一个用于选择房间价格；再添加一个按钮控件，用于获取用户提交请求；添加一个标签控件，用于显示执行结果。主要代码如下：

```
<table cellspacing="1" cellpadding="3">
<thead><tr><td colspan="2" align="center">添加房间</td></tr></thead>
<tbody>
<tr>
<td class="td_left">房间编号：</td>
<td>
<asp:TextBox ID="txtNumber" runat="server" MaxLength="5"></asp:TextBox>
</td>
</tr>
<tr>
<td class="td_left">价格：</td>
<td>
<asp:TextBox ID="txtPrice" runat="server" MaxLength="10"></asp:TextBox>
</td>
</tr>
<tr>
<td class="td_left">房间分类：</td>
<td>
<asp:DropDownList ID="ddlRoomClass" runat="server">
    <asp:ListItem>普通单人间</asp:ListItem>
    <asp:ListItem>普通双人间</asp:ListItem>
    <asp:ListItem>普通海景房</asp:ListItem>
    <asp:ListItem>高级海景房</asp:ListItem>
    <asp:ListItem>豪华海景房</asp:ListItem>
    <asp:ListItem>豪华海景套房</asp:ListItem>
</asp:DropDownList>
</td>
</tr>
<tr>
<td class="td_left">房间状态：</td>
```

```
<td>
<asp:DropDownList ID="ddlStatus" runat="server">
    <asp:ListItem Value="1">正常</asp:ListItem>
    <asp:ListItem Value="2">使用</asp:ListItem>
    <asp:ListItem Value="3">清洁</asp:ListItem>
    <asp:ListItem Value="4">预订</asp:ListItem>
    <asp:ListItem Value="5">锁定</asp:ListItem>
    <asp:ListItem Value="6">维修</asp:ListItem>
</asp:DropDownList>
</td>
</tr>
<tr>
<td colspan="2" align="center">
<asp:Button ID="btnSave" runat="server" Text=" 保存 "
  onclick="btnSave Click" /><br />
<asp:Label ID="lblResultMessage" runat="server" ForeColor="Red"></asp:Label>
</td>
</tr>
</tbody>
</table>
```

这里注意“房间状态”项，我们定义了 6 种房间状态，在数据库里以 int 型存储，但是在页面上我们以相应的文本显示。

(3)　修改“保存”按钮的 onclick 事件处理方法，添加处理程序。代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        //封装房间信息
        Room newRoom = this.GetRoomFromPage();

        DataClassesDataContext dataContext =
            new DataClassesDataContext();    //初始化上下文

        dataContext.Room.InsertOnSubmit(newRoom);    //插入房间信息
        dataContext.SubmitChanges();    //提交更改
        this.lblResultMessage.Text = "保存成功。";
    }
    catch (Exception ex)
    {
        this.lblResultMessage.Text = ex.Message;    //在页面上显示异常信息
    }
}
```

(4)　封装页面房间信息，代码如下：

```
private Room GetRoomFromPage()
{
    /*
```

```
    * 封装页面上的房间信息
    */
    return new Room()
    {
        Number = this.txtNumber.Text,
        Class = this.ddlRoomClass.Text,
        Price = int.Parse(this.txtPrice.Text),
        Status = int.Parse(this.ddlStatus.SelectedValue)
    };
}
```

至此就完成了添加房间信息功能的所有代码。

4) 实现房间信息删除功能

删除房间信息功能需要基于房间信息列表。所以我们直接在房间信息列表页面添加房间信息删除功能。

(1) 首先来修改视图页面。打开 ListRoom.aspx 文件。

(2) 修改 GridView 控件，添加一个模板列，标题为 Delete，并在模板项里添加一个 LinkButton 控件。具体代码如下：

```
<asp:GridView ID="gvRoom" runat="server">
    <Columns>
        <asp:TemplateField HeaderText="Delete">
            <ItemTemplate>
                <asp:LinkButton ID="lnkbtnDelete" runat="server" Text="删除"
                 OnClientClick="return confirm('是否删除?')"
                 CommandArgument='<%# Eval("Number") %>'
                 OnCommand="lnkbtnDelete_Command"></asp:LinkButton>
            </ItemTemplate>
        </asp:TemplateField>
    </Columns>
</asp:GridView>
```

(3) 在页面后台代码 ListRoom.aspx.cs 文件里添加 LinkButton 的 OnCommand 事件处理程序。代码如下：

```
protected void lnkbtnDelete_Command(object sender, CommandEventArgs e)
{
    try
    {
        DataClassesDataContext dataContext =
          new DataClassesDataContext();    //初始化数据上下文
        /* 删除指定编号的房间信息 */
        dataContext.Room.DeleteOnSubmit(
            dataContext.Room
            .Single(r => r.Number == e.CommandArgument.ToString()));
        dataContext.SubmitChanges();  //提交更改

        this.BindRoomList(); //重新绑定列表
```

```
    }
    catch { }
}
```

这样就完成了对房间信息的删除功能。

5)　实现房间信息的修改功能

修改房间信息功能也需要由列表页面支持。我们可以让用户在列表页面选择要修改的记录，跳转到修改编辑页面。

(1)　先修改列表页面。打开 ListRoom.aspx 页面。

(2)　修改 GridView 控件，添加一个模版列，标题为 Edit，模版项里是一个超级链接，链接到编辑页面，并将记录 ID 以 GET 方式传递到编辑页面。代码如下：

```
<asp:TemplateField HeaderText="Edit">
    <ItemTemplate>
    <a href='<%# "EditRoom.aspx?id=" + Eval("id") %>'>修改</a>
    </ItemTemplate>
</asp:TemplateField>
```

以上是新加的模板列的代码，我们可以把它添加到 GridView 控件的 Columns 标签下，与已添加的 Delete 模板列同级。

(3)　保存并关掉 ListRoom.aspx 页面，我们再来添加一个 Web 页面，命名为 EditRoom.aspx。视图代码与前面 AddRoom.aspx 页面的视图代码一样，直接复制过来即可。

(4)　要修改，就先把原始的信息展示给用户，让用户自行选择要修改的项。所以我们要先将原始数据填充到页面视图：

```
private void FillPage(Room room)
{
    /*
    * 填充页面控件内容
    */
    this.txtNumber.Text = room.Number;
    this.txtPrice.Text = room.Price.ToString();
    this.ddlRoomClass.SelectedValue = room.Class;
    this.ddlStatus.SelectedValue = room.Status.ToString();
}
```

这是填充页面视图的代码，我们一会儿取出数据后调用它填充到页面里。

BindRoomInfo()方法的代码如下：

```
private void BindRoomInfo()
{
    try
    {
        int id = int.Parse(Request["id"]);    //获取指定的房间 ID
        DataClassesDataContext dataContext =
          new DataClassesDataContext();    //初始化数据上下文
        Room room = dataContext.Room.Single(r => r.id == id);    //获取房间信息
```

```
        this.FillPage(room);    //填充信息
    }
    catch { }
}
```

好了，我们完成了数据的读取，填充工作。现在需要在页面首次载入的时候执行该操作。所以这里修改 Page_Load 方法，修改后的代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        this.BindRoomInfo();
    }
}
```

(5) 至此完成了初始数据填充的功能，下面需要在单击“保存”按钮的时候执行修改操作。我们修改“保存”按钮的单击事件处理程序，添加相应的操作。代码如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        int id = int.Parse(Request["id"]);    //获取指定的房间 ID
        DataClassesDataContext dataContext =
          new DataClassesDataContext();    //初始化数据上下文
        Room room = dataContext.Room.Single(r => r.id == id);  //查询得到房间对象
        /* 修改对象信息 */
        room.Class = this.ddlRoomClass.SelectedValue;
        room.Status = int.Parse(this.ddlStatus.SelectedValue);
        room.Price = int.Parse(this.txtPrice.Text);
        room.Number = this.txtNumber.Text;

        dataContext.SubmitChanges();     //提交更改
        this.lblResultMessage.Text = "保存成功！";
    }
    catch (Exception ex)
    {
        this.lblResultMessage.Text = "出现异常，异常信息：" + ex.Message;
    }
}
```

这样就完成了修改房间信息的所有代码。

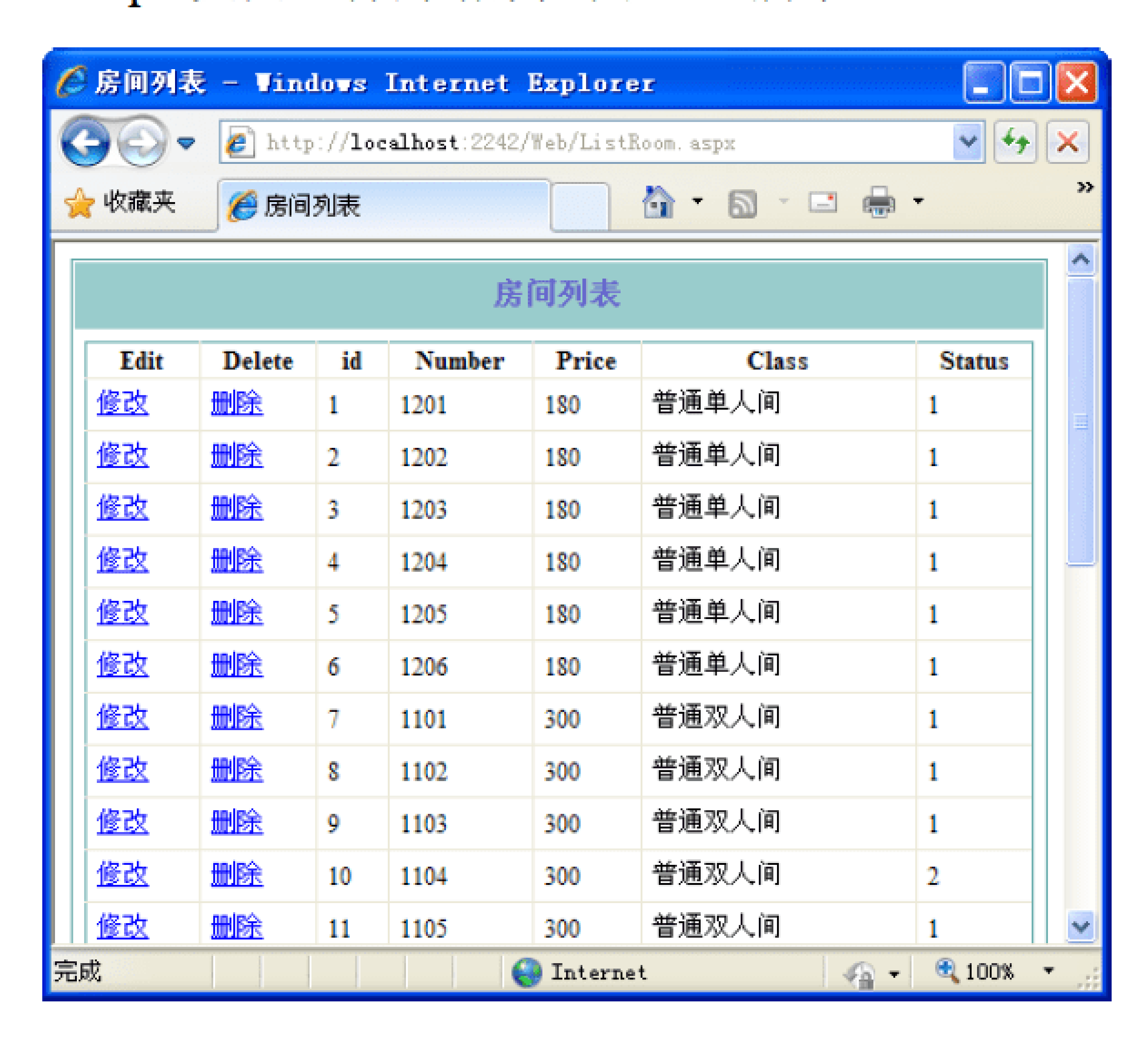
这个实例做到现在，我们完成了对房间信息管理的所有操作。

可以发现整个过程几乎都是在实现我们的业务逻辑，基本上都没去考虑怎么对数据库进行操作。而且，这个实例中因为我们省去了繁琐的数据库操作，程序代码大大地缩短了，而且使程序结构看起来更为清晰了。

5.6.4　运行结果

完成了所有操作之后，我们来运行一下这个程序。

先来访问 ListRoom.aspx 页面，访问结果如图 5-17 所示。



Edit	Delete	id	Number	Price	Class	Status
修改	删除	1	1201	180	普通单人间	1
修改	删除	2	1202	180	普通单人间	1
修改	删除	3	1203	180	普通单人间	1
修改	删除	4	1204	180	普通单人间	1
修改	删除	5	1205	180	普通单人间	1
修改	删除	6	1206	180	普通单人间	1
修改	删除	7	1101	300	普通双人间	1
修改	删除	8	1102	300	普通双人间	1
修改	删除	9	1103	300	普通双人间	1
修改	删除	10	1104	300	普通双人间	2
修改	删除	11	1105	300	普通双人间	1

图 5-17　房间列表

因为房间 1201 经过再次装修，改为普通双人间的规格。这里我们要把 1201 号房间修改为“普通双人间”，价格也调整为 300 元。

我们在列表页单击 Number 列为 1201 的记录行的“修改”链接，来到修改页面。修改相应的项以后单击保存，提示“保存成功！”，如图 5-18 所示。

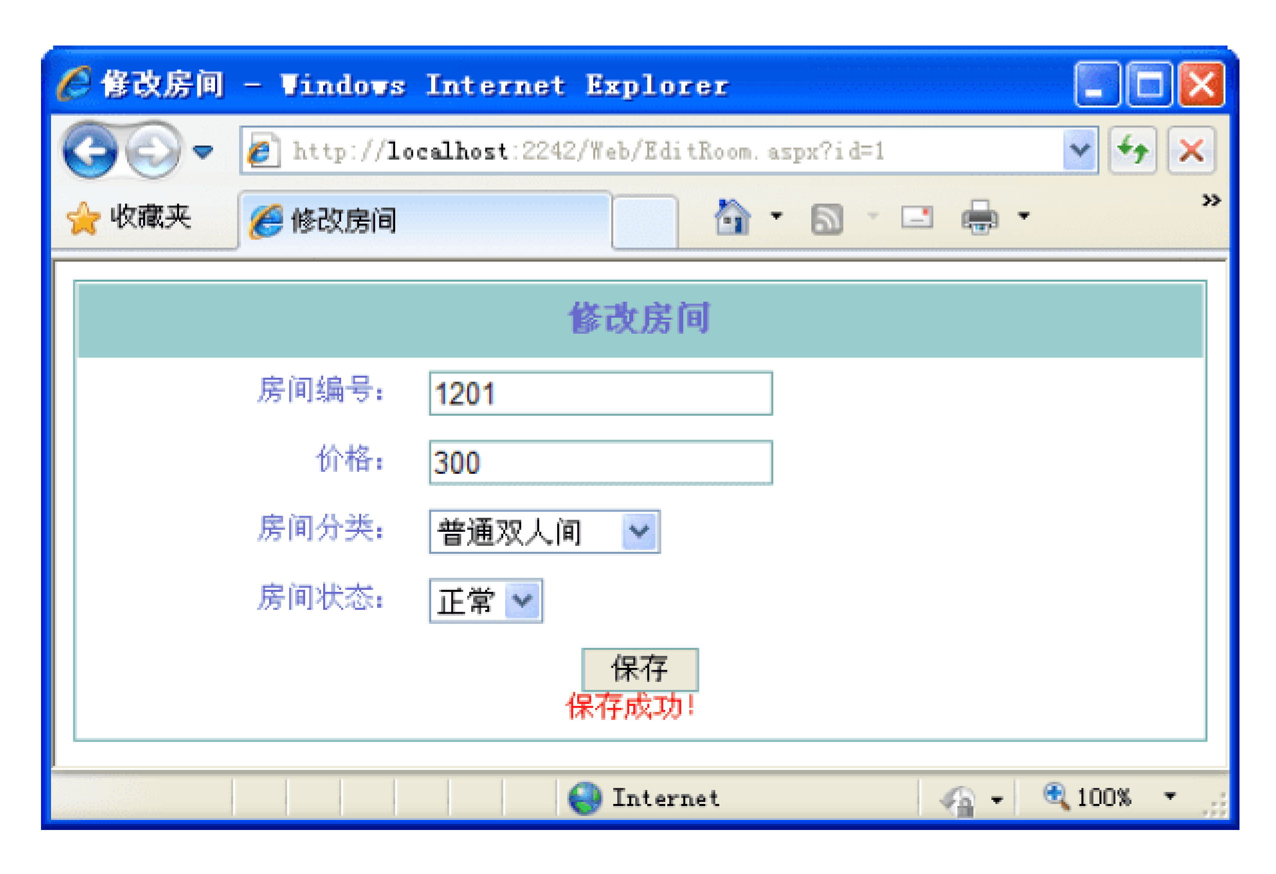
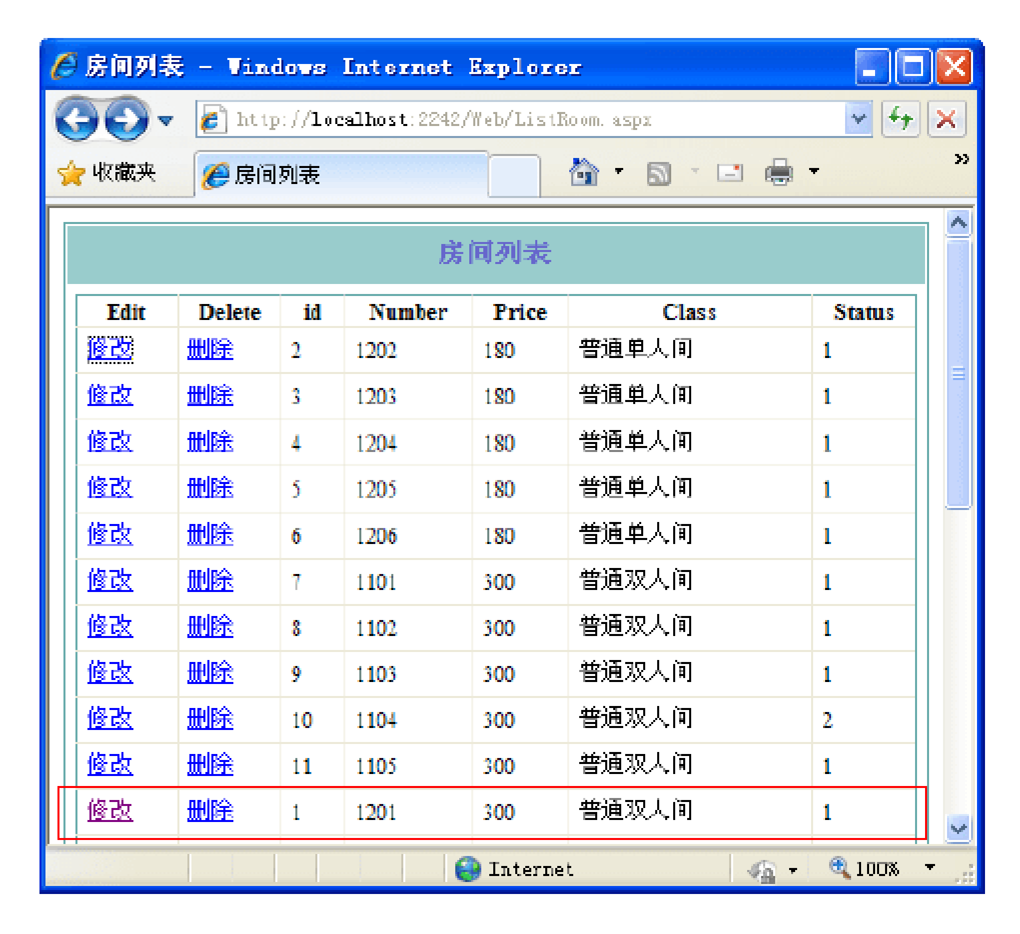


图 5-18　修改房间信息

重新访问 ListRoom.aspx 页面，可以看到 1201 号房间已经修改成功了，如图 5-19 所示。



Edit	Delete	id	Number	Price	Class	Status
修改	删除	2	1202	180	普通单人间	1
修改	删除	3	1203	180	普通单人间	1
修改	删除	4	1204	180	普通单人间	1
修改	删除	5	1205	180	普通单人间	1
修改	删除	6	1206	180	普通单人间	1
修改	删除	7	1101	300	普通双人间	1
修改	删除	8	1102	300	普通双人间	1
修改	删除	9	1103	300	普通双人间	1
修改	删除	10	1104	300	普通双人间	2
修改	删除	11	1105	300	普通双人间	1
修改	删除	1	1201	300	普通双人间	1

图 5-19　修改记录后的房间列表

又经过一次装修，本来是员工休息室的 1207 号房间也改为普通单人间。所以我们要把 1207 号房间的信息添加到数据库中。我们访问 AddRoom.aspx 页面，录入相应信息，单击“保存”按钮，提示添加成功，如图 5-20 所示。

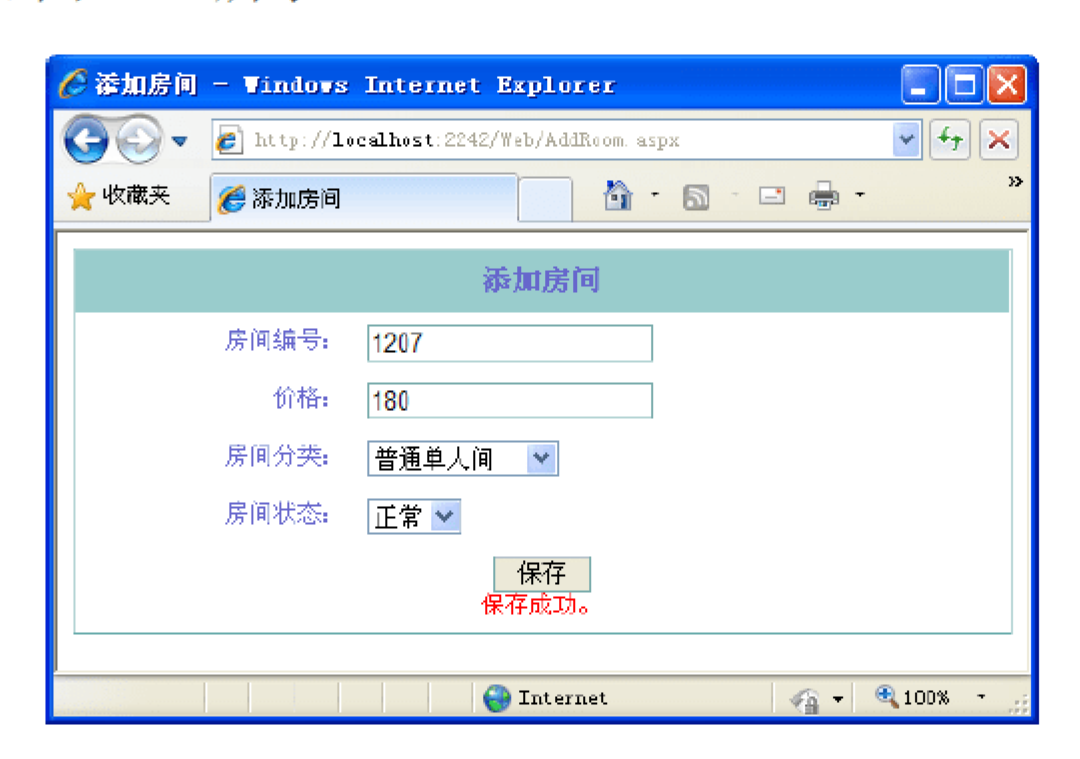


图 5-20　添加房间信息

再次访问 ListRoom.aspx 页面，发现新的记录确实已经保存成功，如图 5-21 所示。



Edit	Delete	id	Number	Price	Class	Status
修改	删除	2	1202	180	普通单人间	1
修改	删除	3	1203	180	普通单人间	1
修改	删除	4	1204	180	普通单人间	1
修改	删除	5	1205	180	普通单人间	1
修改	删除	6	1206	180	普通单人间	1
修改	删除	27	1207	180	普通单人间	1
修改	删除	1	1201	300	普通双人间	1
修改	删除	7	1101	300	普通双人间	1
修改	删除	8	1102	300	普通双人间	1

图 5-21　添加记录后的房间列表

后来因为 1207 号房间改为客房，员工休息极不方便，所以酒店又决定将 1207 号房间改回员工休息室。所以我们要把 1207 号房间的信息从数据库中删除。

先在房间列表页面找着 Number 列值为 1207 的记录行，单击该行的“删除”按钮。系统提示“是否删除？”，单击“确定”按钮即可将该房间号删除，如图 5-22 和图 5-23 所示。

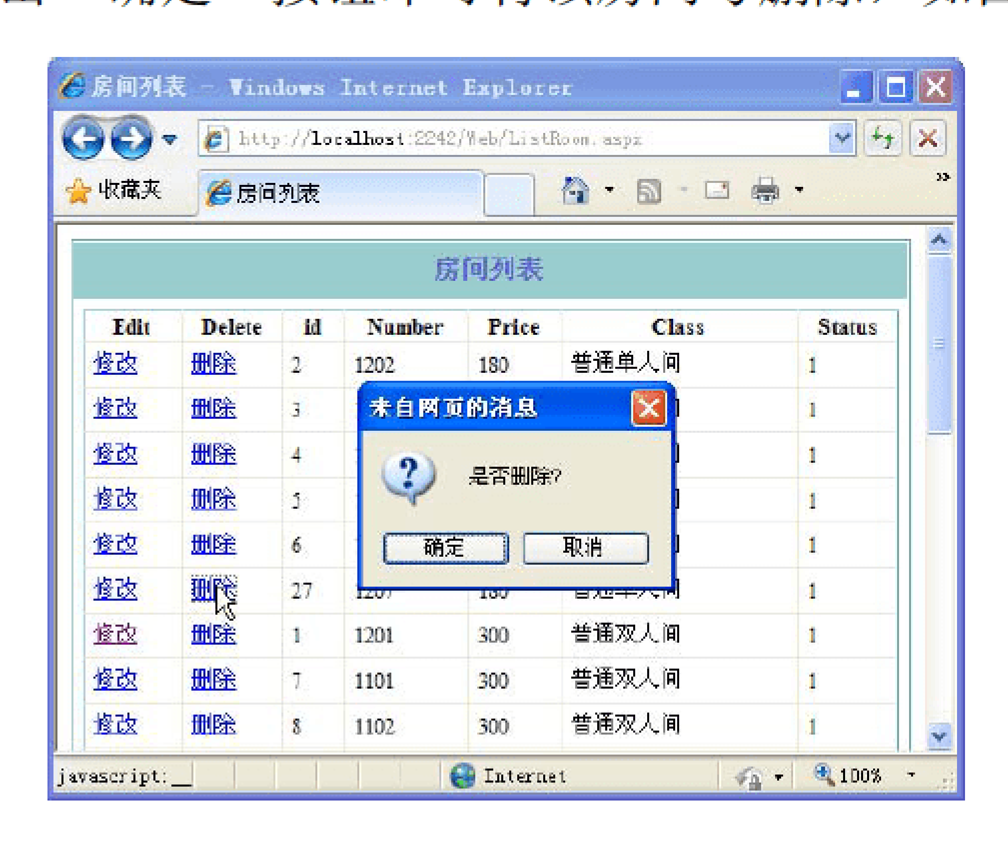


图 5-22　删除房间信息

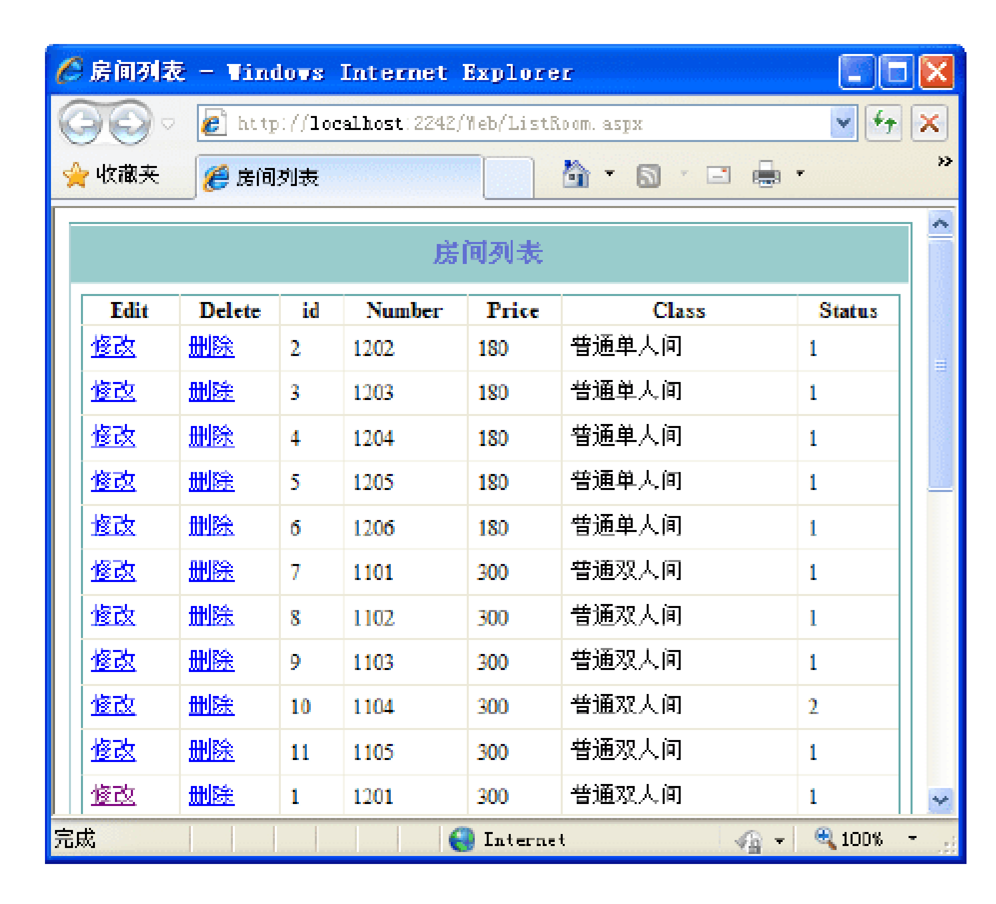


图 5-23　删除后的房间列表

5.6.5　实例分析



源码解析：

实例先创建一个 DataClasses.dbml 文件来构建 Linq to SQL 的 ORM 环境。

在查询的时候我们使用 Linq 语句操作数据上下文对象的 Room 属性，并将查询结果绑定到数据控件的 DataSource 属性。

在实现添加功能时，我们将封装过的数据对象作为参数调用数据上下文对象的 Room 属性的 InsertOnSubmit 方法，最后执行数据上下文对象的 SubmitChanges 方法来提交添加请求。

在删除房间信息时，使用数据上下文对象的 Room 属性的 Single 方法查询出该房间信息，并将查询结果作为参数执行数据上下文对象的 Room 的 DeleteOnSubmit 方法，最后执行数据上

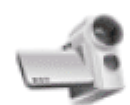
下文对象的 SubmitChanges 方法来提交添加请求。

执行修改功能时，我们需要先使用 Single 方法查询出相应的数据对象，修改查询出来的数据对象的值，然后执行数据上下文对象的 SubmitChanges 方法来提交添加请求。

5.7 把对象直接存储到文件

我们知道直接操作文件需要用到数据流，而.NET 提供了一种能将对象状态直接保存到数据流中的操作，那就是序列化(Serializable，也叫串行化)。

在分布式网络通信技术中(例如远程处理或者 Web Service)，常常要把对象通过网络流进行传输。在单机系统要保存一些没有必要保存到数据库里的状态信息的时候，通常我们会嫌自己写程序格式化对象状态信息比较麻烦，也常常会用序列化的方式直接保存对象到文件，这样我们在下次使用的时候就可以直接把文件内容恢复成对象。

 **视频教学：**光盘/videos/5/Serializable.avi　　长度：5 分钟

5.7.1 基础知识——序列化

在.NET Framework 中，可以通过序列化直接把对象保存到磁盘或其他存储介质中。同样，.NET Framework 当然也提供了一些序列化程序来完成这项任务。

.NET Framework 提供了两种序列化技术：二进制与 XML。这两种技术有很多不同，最主要的区别是 XML 是一个可读的，开放的通用标准，而二进制则不是。

- 二进制：二进制方式序列化会将类的所有字段与属性序列化，并保持类型保真度。我们可以在不同的应用程序间共享对象，也可以将对象序列化到磁盘等存储介质上。
- XML：这种方式只序列化公共的属性和字段。因为是基于通用标准的 XML 文本文件的形式存在，所以不保持类型的保真度。

接下来介绍一下如何将对象序列化到磁盘文件中然后反序列化该对象。

.NET Framework 为了保证数据的安全性，并不允许所有的对象都可序列化。要序列化某对象，需要将该对象声明为可序列化。

使对象可序列化时，可将对象所属的类标记为可序列化的，也可以使该类实现 ISerializable 接口，并自行序列化指定的字段。

相对来说，肯定是第一种比较方便。我们并没有特殊要求，就使用第一种方式实现可序列化实例的类。

把一个类标记为可序列化的方法非常简单，格式为：

```
[Serializable]
class LoginObject
{
   /* Class Body */
}
```

要序列化对象，我们需要使用 System.Runtime.Serialization.Formatters.Binary 命名空间下的 BinaryFormatter 类。该类提供了 Serialize 和 Deserialize 两个方法实现序列化和反序列化。

当然，序列化和反序列化都要基于对“流”的操作。这里我们要序列化对象到磁盘文件，就要用到 System.IO 命名空间下的 FileStream 类。

好了，万事俱备，只欠东风，可以开始干活了。

5.7.2　实例描述

假设我们想要记录一下系统最后一次用户登录的信息，其中包括登录用户名、登录时间、登录 IP。记录以后在每次打开登录页面时显示一下上次登录的信息。

可能这个需求有点过分了，不过假设我们就遇上一个这样的客户，假设我们的客户就这么过分。没招，做吧！又不是不会。

仅仅只要一条信息，就要在数据库中建个表？有点太浪费了吧，可以保存到文件中。

5.7.3　实例应用

【例 5-11】把对象直接存储到文件。

我们要模拟一下登录效果，就在上节用的项目中创建一个登录页面吧。

(1)　打开项目，新建一个 Web 页面，命名为 Login.aspx。

(2)　打开 Login.aspx，在视图里添加登录名和密码两个文本框，接收用户输入，再添加一个提交的命令按钮，再添加 3 个显示上次登录信息的标签。代码如下：

```
用户名：<asp:TextBox ID="txtLoginName" runat="server"></asp:TextBox><br />
密　码：<asp:TextBox ID="txtPassword" runat="server"
TextMode="Password"></asp:TextBox><br />
<asp:Button ID="btnLogin" runat="server" Text=" 登录 "
onclick="btnLogin_Click" /><br />
<br />
<h4>上次登录信息</h4>
用户名：<asp:Label ID="lblLoginName" runat="server"></asp:Label><br />
登录时间：<asp:Label ID="lblLoginTime" runat="server"></asp:Label><br />
登录 IP：<asp:Label ID="lblLoginIP" runat="server"></asp:Label>
```

(3)　要序列化登录信息对象，先得建一个登录信息的类，并标记为可序列化，代码如下：

```
[Serializable]
class LoginObject
{
    public string LoginName { get; set; }
    public DateTime LoginTime { get; set; }
    public string LoginIP { get; set; }
}
```

(4)　在 Login.aspx.cs 文件里添加两个用于序列化和反序列化 LoginObject 对象的方法，代

码如下：

```
private void Serializable(string fileName, LoginObject lo)
{
    /*
    * 序列化对象到指定文件
    */

    BinaryFormatter bf = new BinaryFormatter();    //创建二进制格式对象
    /* 序列化到文件中 */
    using (FileStream fs = new FileStream(fileName, FileMode.OpenOrCreate))
    {
        bf.Serialize(fs, lo);
    }
}

private LoginObject Deserializable(string fileName)
{
    /*
    * 反序列化对象
    */
    LoginObject lo = null;    //声明一个登录信息引用
    BinaryFormatter bf = new BinaryFormatter();    //创建二进制格式对象
    /* 反序列化对象 */
    using (FileStream fs = new FileStream(fileName, FileMode.Open))
    {
        lo = bf.Deserialize(fs) as LoginObject;
    }
    return lo;
}
```

(5) 修改登录按钮的事件处理程序，封装对象并执行序列化操作。代码如下：

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    /* 初始化文件名称 */
    string fileName = Server.MapPath("~/") + "lastLogin.txt";
    /* 封装登录信息对象 */
    LoginObject lo = new LoginObject()
    {
        LoginName = this.txtLoginName.Text,
        LoginTime = DateTime.Now,
        LoginIP = Request.UserHostAddress
    };
    this.Serializable(fileName, lo);    //序列化对象到指定文件
}
```

(6) 序列化完了，还需要在页面加载的时候将上次序列化的信息展示到页面上，我们来修改 Page_Load 方法，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    string fileName = Server.MapPath("~/") + "lastLogin.txt";

    try
    {
        /* 反序列化对象 */
        LoginObject lo = this.Deserializable(fileName);
        /* 将反序列化的值显示到页面 */
        this.lblLoginIP.Text = lo.LoginIP;
        this.lblLoginName.Text = lo.LoginName;
        this.lblLoginTime.Text = lo.LoginTime.ToString();
    }
    catch { }

}
```

至此就完成了本实例的所有编码工作。

5.7.4　运行结果

重新生成项目，然后打开浏览器访问 Login.aspx 页面，如图 5-24 所示。



图 5-24　登录页面

打开的时候页面上什么都没有，因为还没有序列化过的信息。

这个时候我们输入用户名和密码(密码无所谓)，单击“登录”按钮。页面的信息还是没什么变化。不过实际上这次登录信息已经序列化到指定的文件中了。

我们到项目的根目录下用记事本打开该文件，可以看到乱七八糟的一堆字符，如图 5-25 所示。

在这个文件里我们依稀可以看到 LoginName、LoginObject 之类的词组，甚至还有我们输入的内容 joker。可见以序列化方式保存对象并不安全，所以不要随意保存机密的东西。

此时切换到浏览器，刷新一下，可以看到上次登录的信息已经反序列化成功了，如图 5-26 所示。

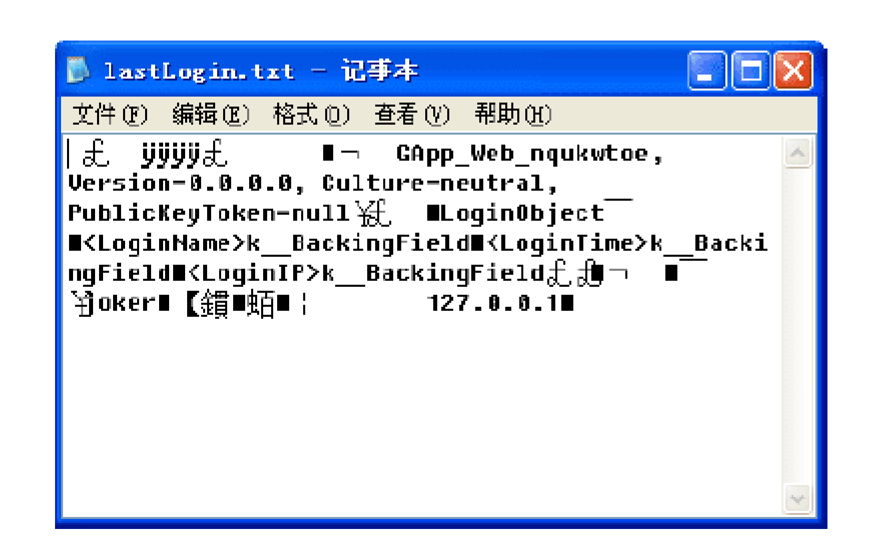


图 5-25　登录信息序列化结果



图 5-26　反序列化成功

5.7.5　实例分析



源码解析：

在实例中我们要用到 BinaryFormatter 对象和 FileStream 对象。

在执行序列化的时候需要使用 BinaryFormatter 对象的 Serialize 方法来将指定实体对象实例化到指定流中。

在执行反序列化操作时需要使用 BinaryFormatter 对象的 Deserialize 方法将指定流中的对象序列化成实体。然后使用 as 关键字将实体对象转换成我们想要的类型。

5.8　常见问题解答

5.8.1　怎么让C#与SQL Server数据库连接



怎么让 C#与 SQL Server 数据库连接？

网络课堂：http://bbs.itzcn.com/thread-3259-1-1.html

我是 ASP.NET 初学者，现在需要做一程序连接到 SQL Server 上，给我说简单一点，不过最好详细一点！

【解决办法】：嗯……超级简单，7 个步骤即可搞定。

(1) 导入命名空间，语句如下：

```
using System.Data.SqlClient;                    //连接 SQL Server 数据库专用命名空间
```

(2) 创建连接，语句如下：

```
SqlConnection lo_conn = New SqlConnection(
  "Server=服务器名字或 IP;Database=数据库名字;uid=用户名;pwd=密码");
```

(3) 打开连接(注意第 2 步并没有真正连接数据库)，语句如下：

```
lo_conn.Open();  //真正与数据库连接
```

(4) 向数据库发送 SQL 命令要使用 SqlCommand，语句如下：

```
SqlCommand lo_cmd = new SqlCommand();    //创建命令对象
lo_cmd.CommandText = "这里是 SQL 语句";   //写 SQL 语句
lo_cmd.Connection = lo_con;              //指定连接对象，即上面创建的
```

(5) 处理 SQL 命令或返回结果集，语句如下：

```
lo_cmd.ExecuteNonQuery();
  //这仅仅执行 SQL 命令，不返回结果集，实用于建表、批量更新等不需要返回结果的操作
SqlDataReader lo_reader = lo_cmd.ExecuteReader();         //返回结果集
```

(6) 以数据集的方式返回结果集，语句如下：

```
SqlDataAdapter dbAdapter = new SqlDataAdapter(lo_cmd);  //注意与上面的区分开
DataSet ds = new DataSet();                             //创建数据集对象
dbAdapter.Fill(ds);  //用返回的结果集填充数据集，可以被能操作数据的控件 DataBind
```

(7) 关闭连接，语句如下：

```
lo_conn.Close();
```

5.8.2 SqlDataReader的Read()方法读取的值是什么类型



SqlDataReader 的 Read()方法读取的值是什么类型？

网络课堂：http://bbs.itzcn.com/thread-3267-1-1.html

看如下代码：

```
SqlDataReader reader = DBHelper.GetDataReader(sql, param);
if(reader.Read())
{
   string aa= (string)reader["U_name"];
}
```

这里对 reader["U_name"]我不想做强制转换(string)，想把它作为参数，可是不知道是什么类型的？因为有时候数据库里是 NULL 值的话它就强转报错，所以我要判断，想写个公有方法，可不知道 reader["U_name"]是什么类型的？谁能告诉我啊！谢谢了。

【解决办法】：这很简单，reader[“U_name”]是 Object 类型，以后要记住了啊。

5.8.3 Parameter对象的怪事



Parameter 对象的怪事。
网络课堂：http://bbs.itzcn.com/thread-3289-1-1.html

看如下代码：

```
dim cmd, param
Set cmd = Server.CreateObject("ADODB.Command")
cmd.ActiveConnection = session("cnn")
cmd.CommandText = "oilperkm"
cmd.CommandType = 4
cmd.Prepared = True
```

现在要创建 Parameter 对象，将数据追加到 Parameters 数据集合中：

```
Set param = cmd.CreateParameter("startDate", 200, 1)   '——这句报错!
cmd.Parameters.Append param
```

报错信息：

```
错误类型：ADODB.Parameters (0x800A0E7C)
```

我这个语法有问题吗？不会吧！看别人都这样写的。

【解决办法】：这句当然有错，CreateParameter 有 5 个参数，你只写了 3 个，当然，输出就 4 个。示范：

```
Set param = cmd.CreateParameter("startDate", 200, adParamInput, 数据长度, 变量)
'或者:
Set param = cmd.CreateParameter("startDate", 200, adParamInput, 数据长度)
cmd.Parameters("startDate") = 变量
```

5.9 习　　题

一、填空题

(1) SqlCommand 对象的____________属性可以设置 SqlCommand 对象所使用的数据库连接。

(2) SqlConnection 对象在使用完的时候必须关闭以释放资源，关闭对象需要使用__________方法来完成。

(3) 如果 SqlCommand 类的对象执行的是存储过程，需要将该对象的 CommandType 属性设置为__________。

(4) SqlDataReader 类的实例使用索引器获取到的数据是__________类型。

(5) FileStream 类是在__________命名空间下定义的。

(6) 补充以下代码段中的空白:

```
...
using (SqlConnection conn = new SqlConnection(connStr))
{
    SqlCommand cmd = new SqlCommand(sql, conn);
    conn.Open();
    SqlDataReader reader = cmd.ExecuteReader();
    while (__________)
    {
        /* 封装数据代码省略 */
    }
    reader.Close();
}
...
```

二、选择题

(1) 在.NET 框架类库中，所有与多线程机制应用相关的类都放在__________命名空间中。

A. System.DB　　B. System.Data

C. System.DataBase　　D. System.ADO

(2) 在 ADO.NET 中，数据适配器用于在__________之间交换数据。

A. 数据源和数据源　　B. 数据集和数据集

C. 数据源和数据集　　D. 数据源和数据集或数据集与数据集

(3) 在 ADO.NET 中，默认提供了__________等几种数据库的访问。(多选)

A. SQL Server　　B. Oracle

C. MySQL　　D. ODBC

(4) Command 对象 ExecuteNonQuery 方法的功能是__________。

A. 执行一个空查询　　B. 返回 SQL 语句影响行数

C. 执行一个存储过程　　D. 返回一个集合

(5) 在 Linq to SQL 中，每执行一次添加、修改或删除之类的编辑数据库的操作，都需要执行一下 DataClassesDataContext 类的__________方法。

A. SubmitChanges　　B. Submit

C. Single　　D. ChangeOnSubmit

(6) 将一个对象序列化到流中需要用到__________。

A. 被序列化对象的 ToString 方法

B. FileStream 类的 Read 方法

C. BinaryFormatter 类的 Deserialize 方法

D. BinaryFormatter 类的 Serialize 方法

三、上机练习

上机练习：聊天室。

使用 ADO.NET 实现一个简单的聊天室功能。聊天室可以发表言论，在页面上列出最新的 20 条留言信息，界面如图 5-27 所示。

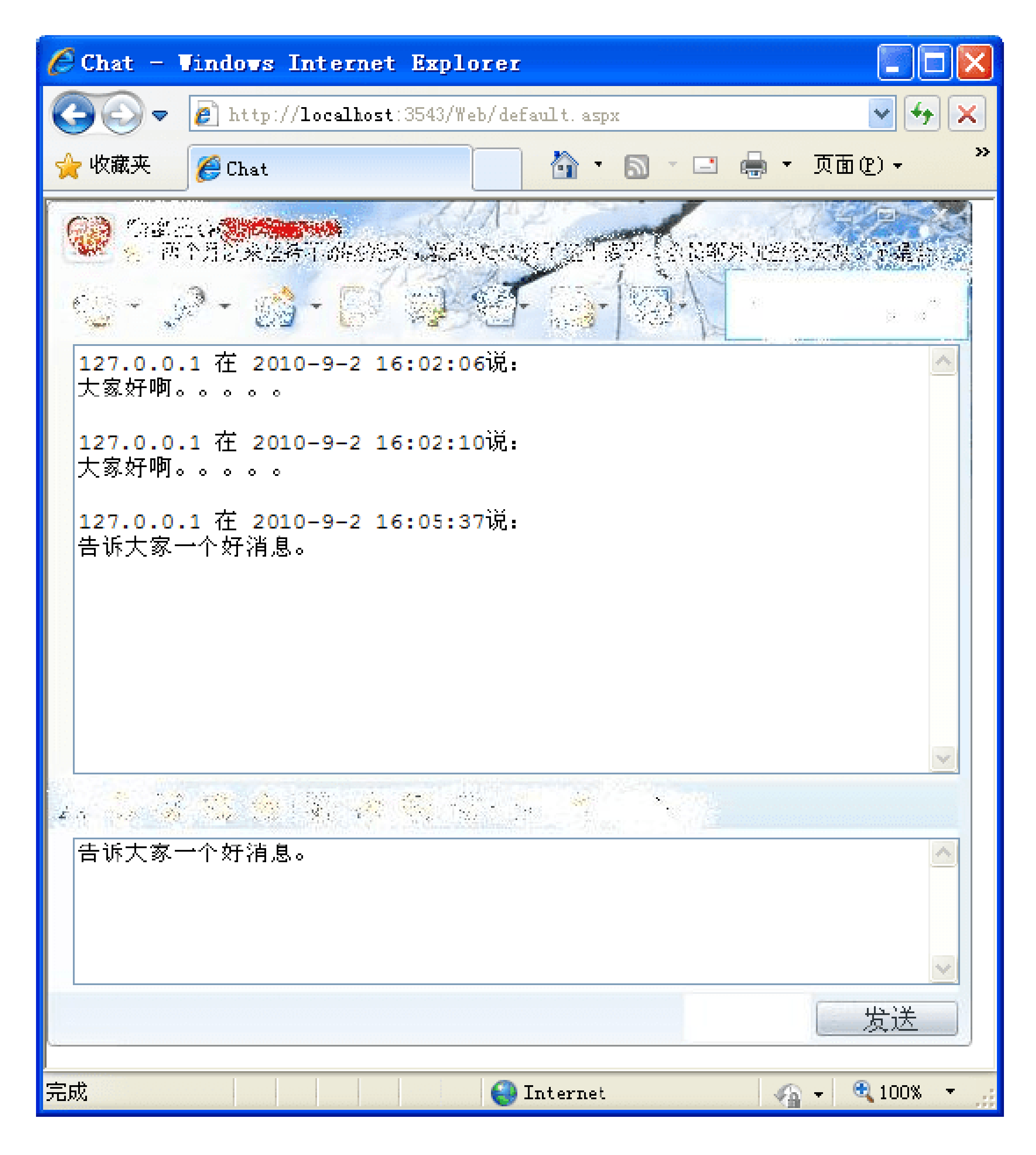


图 5-27　在线聊天



第 6 章　优雅的数据展示

内容摘要：

在一般的 Web 应用程序中，经常需要把数据库里的信息展示给用户。

在传统的 Web 应用程序中(如 ASP)，每次展示的时候我们都要在页面编写复杂的循环代码列表展示数据集合，或者将一个对象的数个属性信息通过编写代码一个个地展示到页面上。

这样很费事、容易出错，而且复杂啰嗦的代码很难维护和修改。

ASP.NET 为我们提供了一系列的服务器端控件，使 Web 应用程序开发人员能很轻松地在页面上展示信息。

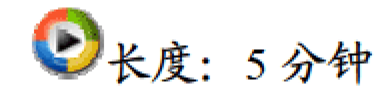
学习目标：

- 熟练使用 Repeater 控件展示列表
- 熟练使用 DataList 控件展示列表
- 熟练使用 GridView 控件展示列表
- 熟练使用 GridView 控件修改记录
- 熟练使用 GridView 控件删除记录
- 熟练使用 DetailsView 控件展示和修改详细内容
- 熟练使用 FormView 展示内容

6.1 自定义格式的博文列表

在创建 Web 应用程序时，我们选择的是“把代码放到单独的文件中”，所以 C#代码都需要写在 cs 结尾的文件中。但是要在页面上展示一个数据列表，就不能再像以前那样直接在页面上写 C#代码了。

不过还好，ASP.NET 提供了一些数据控件，可以很方便地实现列表展示的功能，比如 Repeater 控件。

 视频教学：光盘/videos/6/Repeater.avi 长度：5 分钟

6.1.1 基础知识——Repeater

Repeater 控件是一个纯粹的数据迭代控件，它仅仅封装了页面上的遍历脚本。它的用法非常简单，但是功能却十分强大。

Repeater 控件提供了 5 个项目模板，如表 6-1 所示。

表 6-1 Repeater模板

模 板 名	说 明
HeaderTemplate	头部模板。因为数据列表一般会有表头，为保证代码结构化，表头的内容就可以放在这里
ItemTemplate	项目模板。这里就是普通项的模板，也就是数据列表里每一项在页面上展示的效果就在这里定义
AlternatingItemTemplate	交替项模板。对应 ItemTemplate，如果设置该项，该项表示偶数项的模板，一般设置列表奇偶项不同背景色时会用到
SeparatorTemplate	间隔符模板。在每一个 ItemTemplate 或 AlternatingItemTemplate 项之间插入的分隔用的内容
FooterTemplate	脚注模板。一般的列表项可能会用脚注说明该列表的信息，这里定义列表脚注的内容

我们来看一下 Repeater 控件的使用方法，示例代码如下：

```
<asp:Repeater ID="Repeater1" runat="server">

<HeaderTemplate><!-- 头部内容 -->
   <div style="background-color:#aaf;">This is heder</div>
</HeaderTemplate>
<ItemTemplate><!-- 项目模板 -->
   <span style="background-color:#eef;">This is item</span>
</ItemTemplate>
<AlternatingItemTemplate><!-- 交替项 -->
```

```
    <span style="background-color:#ddf;">This is Alternating Item</span>
</AlternatingItemTemplate>
<SeparatorTemplate><!-- 项目分隔符 --><br /></SeparatorTemplate>
<FooterTemplate><!-- 脚部内容 -->
    <div style="background-color:#dde;">This is Footer</div>
</FooterTemplate>

</asp:Repeater>
```

再来看一下它在设计视图的预览效果，便可以清楚地了解各个模板的用途。预览如图 6-1 所示。

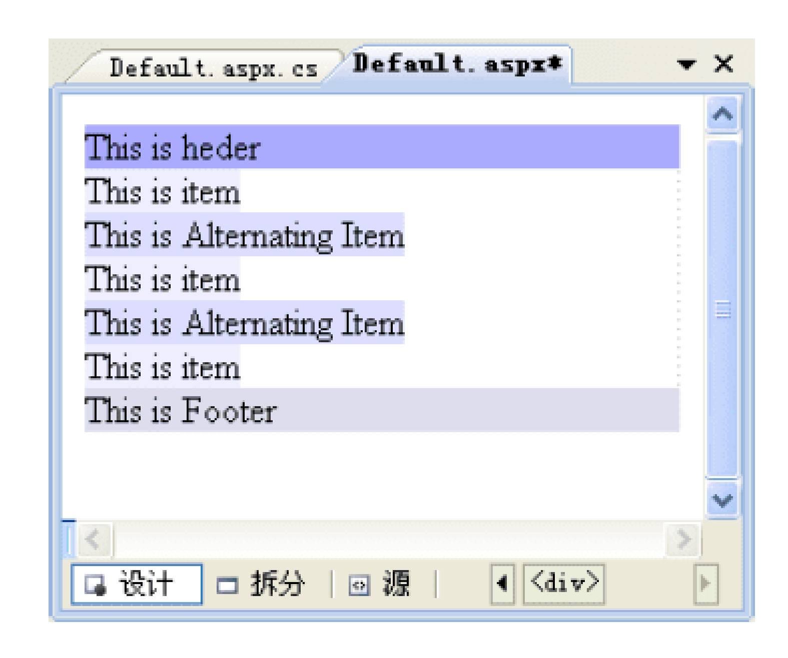


图 6-1　Repeater预览效果

应用的时候，我们可以对 Repeater 控件绑定数据源控件，也可以编写代码为 Repeater 对象的 DataSource 属性指定一数据集合，并调用 DataBind 方法重新生成列表。

在 Repeater 对象的模板里，我们可以使用数据绑定表达式<%# Eval(“PrameterName”) %>或<%# Bind(“PrameterName”) %>将列表中每一项的对应字段展示到页面上。

例如要列出博客文章的标题和发布时间，可以这样编写 ItemTemplate：

```
<ItemTemplate>
    <%# Eval("Title") %>    <%# Bind("PublishTime") %>
</ItemTemplate>
```



Eval 和 Bind 在使用方法上并没有什么区别。不过在以后要讲的几个可编辑的控件上，被编辑的字段就必须使用 Bind 方法。具体原因以后再详细介绍。

这两个方法还可以像 string 类的 Format 方法一样对数据进行格式化。比如格式化日期型数据的格式规则：“年-月-日 时:分:秒”格式化字符串为“{0:yyyy-MM-dd hh:mm:ss}”。

6.1.2　实例描述

闲来没事，注册个域名，开了个空间，想自己写个博客玩玩。

博客程序里当然最主要的就是博客文章了。文章有很多，所以需要用列表控件将它显示在页面上。

觉得 Repeater 比较简单实用，就用它作为数据展示控件。

6.1.3 实例应用

【例 6-1】自定义格式的博文列表。

现在来开发一个自己的博客程序，如果美工不好，可以去网上找一些现成的模板使用。

(1) 首先创建一个 Web 项目。

(2) 在 Default.aspx 页面中，添加一个 Repeater 控件，为实现交替项效果，我们添加了 ItemTemplate 和 AlternatingItemTemplate 两个模板。代码如下：

```
<asp:Repeater ID="rptBlogList" runat="server">
<ItemTemplate>
    <LI>
        <DIV class="postHeader">
            <H3><%# Eval("PublishTime","{0:yyyy-MM-dd}") %></H3>
            <H2><A href="#"><%# Eval("Title") %></A></H2>
        </DIV><DIV class="clear"></DIV>
        <DIV class="postBody">
            <P><%# Eval("Content") %></P>
            <DIV class="clear"></DIV>
        </DIV><DIV class="postFooter">
            <DIV class="menubar"> 发表于
               <SPAN class="time"><%# Eval("PublishTime","{0:hh:mm:ss}") %></SPAN>
            </DIV>
        </DIV>
    </LI>
</ItemTemplate>
<AlternatingItemTemplate>
    <LI style="background-color:#eee;">
        <DIV class="postHeader">
            <H3><%# Eval("PublishTime","{0:yyyy-MM-dd}") %></H3>
            <H2><A href="#"><%# Eval("Title") %></A></H2>
        </DIV><DIV class="clear"></DIV>
        <DIV class="postBody">
            <P><%# Eval("Content") %></P>
            <DIV class="clear"></DIV>
        </DIV><DIV class="postFooter">
            <DIV class="menubar"> 发表于
               <SPAN class="time"><%# Eval("PublishTime","{0:hh:mm:ss}") %></SPAN>
            </DIV>
        </DIV>
    </LI>
</AlternatingItemTemplate>
</asp:Repeater>
```

两个模板内容几乎一样，只是在 AlternatingItemTemplate 模板中的 LI 标记里添加了一个背景颜色的样式。

(3) 在页面后台 Default.aspx.cs 文件里添加列表绑定方法。代码如下：

```
private void BindBlogList()
{
    /*
     * 绑定博客文章列表
     */
    IList<Blog> list = BlogManager.GetList();
    this.rptBlogList.DataSource = list;
    this.rptBlogList.DataBind();
}
```

这里用到了一个 BlogManager 类下的静态方法，用于操作数据库获取一个博客文章列表。具体操作数据库的代码这里就不再啰嗦了。

(4) 我们需要在页面加载的时候执行该方法，所以修改 Page_Load 方法，代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        this.BindBlogList();    //绑定博客列表
    }
}
```

代码完成，来运行一下。

6.1.4　运行结果

生成网站，运行，访问 Default.aspx 页面。效果如图 6-2 所示。

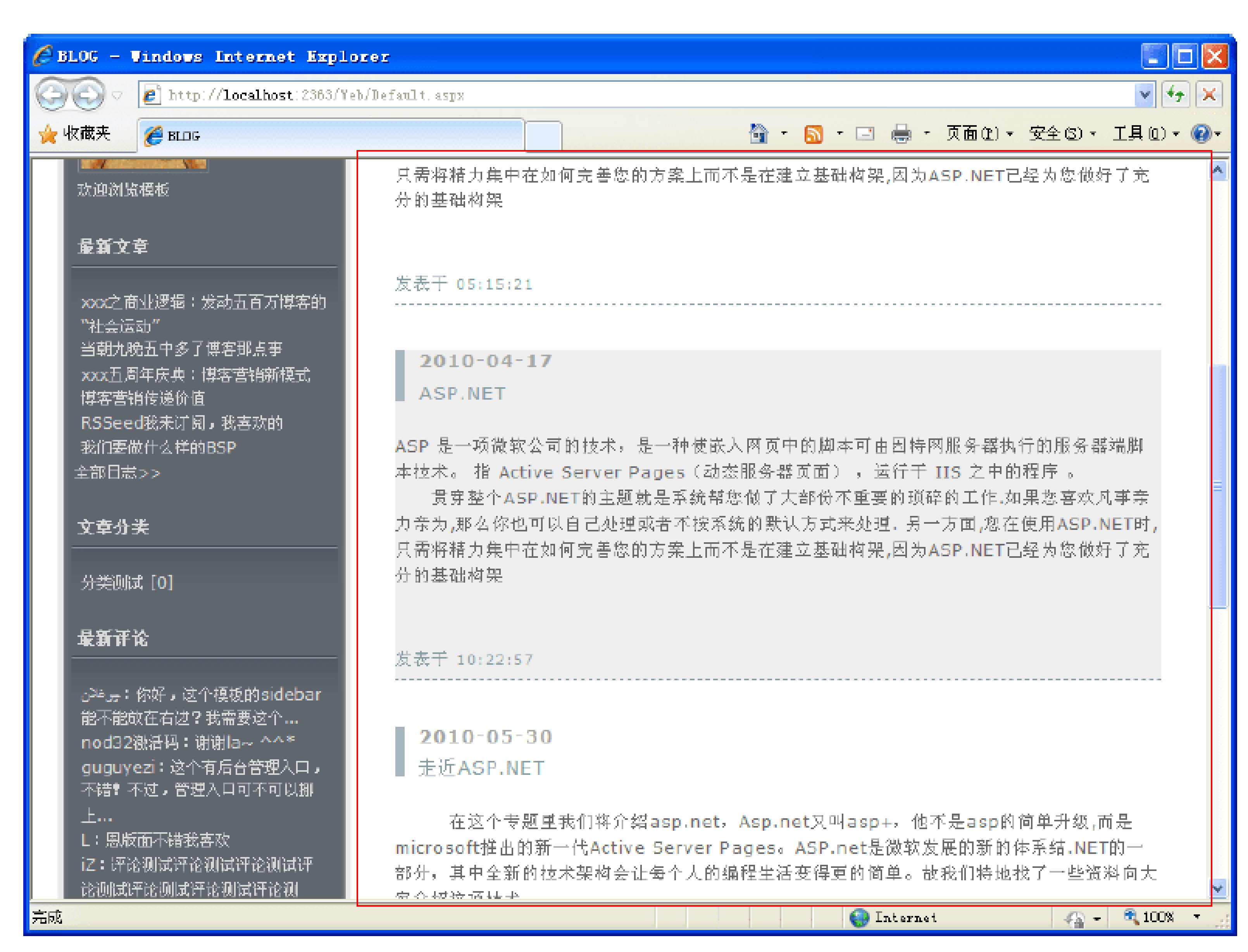


图 6-2　博客列表

图 6-2 中，红色线框内的部分就是我们的 Repeater 生成的内容。可以看到它实现了奇偶项背景交替的效果，更方便用户浏览。

6.1.5 实例分析


源码解析：

实例创建了一个 Repeater 控件，并添加了 ItemTemplate 和 AlternatingItemTemplate 两个模板，添加相应代码实现交替项的效果。

在页面加载的时候取出博客文章列表，设置 Repeater 控件的 DataSource 属性指定数据源，然后调用 Repeater 控件的 DataBind 方法执行绑定操作。

6.2 横排的图片友情链接展示

有时候我们需要在页面上一行行地展示一些项目，比如产品列表，每行显示固定数目，显示数行。

这样可以使用表格来展示数据项，表格的每一个单元格就是一个数据项。

用 Repeater 来实现，完全没有问题。因为 Repeater 生成列表不添加任何扰乱布局的代码，所以我们只要用 HTML 实现以后，将每一项的内容填到 Repeater 控件的项模板里就行了。不过，这样就需要费心编写代码控制 Repeater 每一项的样式。

不过还好，ASP.NET 给我们提供了一个实现该功能的服务器端控件——DataList。


视频教学：光盘/videos/6/DataList.avi　　长度：5 分钟

6.2.1 基础知识——DataList

DataList 称为数据列表控件，DataList 控件和 Repeater 控件一样，也是迭代控件，它能够以事先指定的样式和模板重复显示数据源中的数据，不过它会默认地在数据项目上添加表格来控制页面布局。DataList 控件自动集成了很强大的功能，并提供了多个模板及样式属性供我们使用。

DataList 控件集成了更强大的功能，具体说明如下：

- 支持 7 种模板，并为所有模板提供了相应的样式。
- 能够控制数据项的显示方向，比如横向或纵向显示列表。
- 能控制每一行显示数据的最大数量。
- 提供了对数据选择、编辑、删除等功能。

DataList 控件还有如下两个重要的属性。

- RepeatDirection：项目排列方向。值为 RepeaterDirection 枚举的值，有 Horizontal(横向)和 Vertical(纵向)两项可以选择。

- RepeatColumns：重复列数，显而易见，就是生成的表格每行的个数。



DataList 控件的 RepeatColumns 属性有点意思。它是列数，无论排列方向是横向还是纵向，它都生成这么多列(除非你的项数少于列数)。

DataList 控件除了具有 Repeater 控件所具有的 5 个模板之外，它还具有两个新的模板：EditItemTemplate 和 SelectedItemTemplate。顾名思义，其中一个是编辑项模板，一个是选择项模板，说明它有编辑和选择项的功能。

当然，绑定数据还是使用 Eval 方法和 Bind 方法。不过这里有编辑功能，如果要使用编辑功能，则必须使用 Bind 方法绑定数据，因为只有 Bind 方法可以提交返回值。

6.2.2　实例描述

几乎每个网站都会有友情链接功能。

当然，我们自己的博客为了方便的与朋友交流，也通常会使用友情链接功能来从我们的站点快速地链接到朋友的网站。

一般用的纯文本的友情链接样式，既不突出也不好看，所以很多时候我们会选择使用统一规格的小图片来作为各网站的形象标识。

因为页面显示的是方方正正的图片，所以不能像纯文本一样堆砌显示。为了使页面布局更为美观，我们需要规规整整地把它们放到表格里一排排地显示。

因此，在这里使用 DataList 是最好的选择。

6.2.3　实例应用

【例 6-2】横排的图片友情链接展示。

(1)　打开上一节我们使用的项目。

(2)　在左侧模块底部添加一个 DataList 控件，并加入一个超链接，超链接内空为一个 img 标签。然后分别绑定超链接的 href 属性、图片的 alt 属性和 src 属性。设置 DataList 控件的 RepeaterDirection 属性为 Horizontal、RepeaterColumns 属性为 2、ID 属性为 dlFriendLink。代码如下所示：

```
<asp:DataList ID="dlFriendLink" runat="server" RepeatColumns="2"
  RepeatDirection="Horizontal">
<ItemTemplate>
    <a href='<%# Eval("URL") %>'>
        <img alt='<%# Eval("Title") %>' src='<%# Eval("Picture") %>'
            style="width:85px; height:30px; margin:3px;" />
    </a>
</ItemTemplate>
</asp:DataList>
```

(3)　添加一个绑定友情链接的方法，从数据库得到一个友情链接列表，然后绑定到列表。代码如下：

```
private void BindFriendLinkList()
{
    /*
     * 绑定友情链接列表
     */
    IList<FriendLink> list = FriendLinkManager.GetList();
    this.dlFriendLink.DataSource = list;
    this.dlFriendLink.DataBind();
}
```

这里使用的 FriendLinkManager 类的静态方法 GetList 要从数据源(数据库)中读取友情链接列表。

(4) 当然，还需要在页面加载的时候调用一下。我们需要在 Page_Load 方法的 if 语句的语句块中调用绑定友情链接的方法。代码如下：

```
this.BindFriendLinkList();    //绑定友情链接
```

经过上述编程之后，就可以运行了。

6.2.4 运行结果

运行项目，重新访问 Default.aspx 页面。

运行结果如图 6-3 所示。



图 6-3 友情链接

在图 6-3 中，红线框中的内容就是 DataList 控件生成的内容。

6.2.5　实例分析



源码解析：

该实例创建了一个 DataList 控件，设置 RepeatDirection 属性为 Horizontal 表示横向展示数据，并设置 RepeatColumns 属性的值为 2，表示一数据列表分两列显示。然后添加了一个 ItemTemplate 模板，并设置模板内容。

然后取出友情链接数据列表，设置数据源属性 DataSource，调用绑定方法绑定数据列表。

6.3　快速实现漂亮的博文分类列表

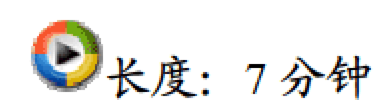
在开发应用程序的时候，我们用得最多的还是以列表的形式展示数据。

在所有 ASP.NET 数据展示控件里，可以说 GridView 控件将 ASP.NET 的设计初衷发挥得最为淋漓尽致。GridView 控件提供了分页展示、排序、删除、修改等功能。VS2008 还为它提供了数个漂亮的格式供用户套用。用户通过简单的设置就能完成上面提到的所有先进功能，而在以前我们做这些工作都需要编写大堆的代码。

GridView 控件在所有数据展示控件里应该是最容易入门和使用的，它很强大，同时也是最难真正用好的。

因为它的强大，我们只需拖入页面一个控件，在页面生成简简单单两行代码就可以将一个数据集合里的字段完全展示出来。同样，因为它的强大，系统为它提供了数十个属性(不包括子属性)和 20 多个事件供用户配置使用。所以作为初学者，想要熟练使用尚需时日。

而且，因为系统对它封装得太多，在实现一些特殊效果时或者做一些高性能应用时，灵活性会大打折扣，现成的一些东西又成累赘了。如何做到恰当地使用，需要我们自己好好把握。

 **视频教学：**光盘/videos/6/GridView.avi　　　长度：7 分钟

6.3.1　基础知识——GridView

GridView 控件又被称为网格视图控件。它和 Repeater 控件和 DataList 控件一样，也是一个迭代控件，可以以表格的形式展示数据。

但同样以表格展示数据，GridView 控件和 DataList 控件却有不同。DataList 控件把每个数据项布局到一个表格中展示，而 GridView 控件把每个数据项作为一个数据行展示，数据项的每个字段就填充在行内的单元格里，展示到页面上。

GridView 为每个字段提供了 7 种字段展示方式，如表 6-2 所示。

因为 GridView 控件能细化到对每一个字段设置格式，所以在上面所说的“模板字段”里我们可以设置 5 个字段模板。另外 GridView 控件还提供了一个空数据模板和一个分页模板，具体说明如表 6-3 所示。

表 6-2 GridView字段

名 称	说 明
BoundField	数据绑定字段，以标签的形式绑定展示数据
CheckBoxField	选择框字段，以复选按钮的形式展示数据，一般用于展示 bool 型数据
HyperLinkField	超链接字段，以超链接的形式展示数据，一般用于执行某些操作或跳转页面的时候使用
ImageField	图像字段，以图像的形式展示数据，这里一般显示缩略图片
ButtonField	按钮字段，以链接按钮的方式展示数据，这里一般用于执行一些自定义命令
CommandField	命令字段，GridView 控件内置命令。有添加、修改、取消、选择、删除等三组选项，前三个是一组，后两个各为一组
TemplateField	模板字段，以自定义模板的方式展示数据

表 6-3 GridView模板

模 板 名	说 明
EmptyDataTemplate	当数据为空时 GridView 控件的显示格式，可别小看，这个用处很大。在数据列表为空时，这个模块总不能显示一片空白吧。设置完这个模板以后，当数据列表为空时就展示这些内容
PagerTemplate	用来设置分页控件的格式
AlternatingItemTemplate	交替项，以前讲过的，这里是针对每一列的交替项格式
EditItemTemplate	编辑项目模板，这里针对该字段的模板，我们可以设置不同的服务器端控件来处理编辑状态下的不同类型数据
FooterTemplate	脚注模板，设置每一列底部的提示内容及格式
HeaderTemplate	头部模板，设置每一列头部的提示内容及格式
ItemTemplate	项目模板，设置列内容的格式

前面提到过，GridView 控件提供了数十个属性，可以详细地对其进行设置以实现各种功能或效果。常用的属性如表 6-4 所示。

表 6-4 GridView属性

属 性 名	说 明
AllowPaging	分页功能是否启用，默认为 false，表示不启用分页功能
AllowSorting	启用排序功能，默认为 false，当该属性为真时用户可以单击某一列的标题以该列为序对列表重新排序。注意这里如果启用，数据源必须为 DataSet、DataView 或 DataTable
DataSource	数据源，object 类型，可以是 DataSet，也可以是集合
DataSourceID	数据源控件 ID，这个属性和 DataSource 属性只能设置一个，否则会报异常
DataKeyNames	数据键名列表，能标识对象记录的字段组合，多个字段将联合作为关键字标识记录
PageIndex	当前页索引，就是当前页码
PageSize	页面大小，即一页展示几条记录
ShowFooter	是否显示脚注

续表

属 性 名	说　明
ShowHeader	是否显示头部
Visible	控件是否显示，几乎所有的服务器对象都有该属性

这里还要注意一点，在使用 DataSource 对 GridView 控件设置值以后，需要调用 DataBind()方法对列表重新绑定，否则数据列表不会更新。

6.3.2　实例描述

在开发 Web 应用程序时，我们总是非常头痛数据的页面展示。有时候只是写一个简单的管理功能，对性能，界面美观都要求不高，但还是不得不写大堆的页面代码，费时费力又不出效果。

就比如现在吧，要对博文列表进行管理，就简简单单的两个字段，要放在以前，我们还得写一堆乱七八糟的代码来实现该功能。

不过既然有了 GridView，就好说多了，我们只需要点点鼠标，一切以往费时费力又不出成绩的编码工作眨眼之间就可以完成了。

6.3.3　实例应用

【例 6-3】快速地实现漂亮的博文分类列表。



这个实例需要有一个数据访问类提供支持。这里数据访问类 BlogClassManager 提供增、删、改、查等公有静态方法。这个实例需要用到返回实体列表的 GetList 方法。

(1) 创建成一个新的 Web 窗体，命名为 EditClassList.aspx。

(2) 在视图页面打开“设计”视图，从左侧工具箱里的“数据”选项卡下找到 GridView 控件，拖一个到 Web 窗体上，在右侧“属性”窗口里修改 ID 属性为 gvClassList，Width 属性为 100%，该控件如图 6-4 所示。

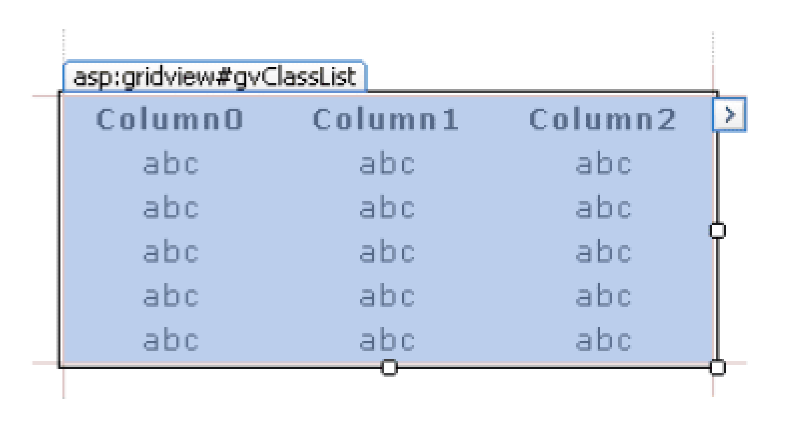


图 6-4　GridView

(3) 这里我们要使用 ObjectDataSource 对 GridView 控件的数据进行绑定。单击 GridView 控件右上角的小方块，在“选择数据源”列表框中选择“<新建数据源...>”。打开“数据源配置向导”对话框，如图 6-5 所示。

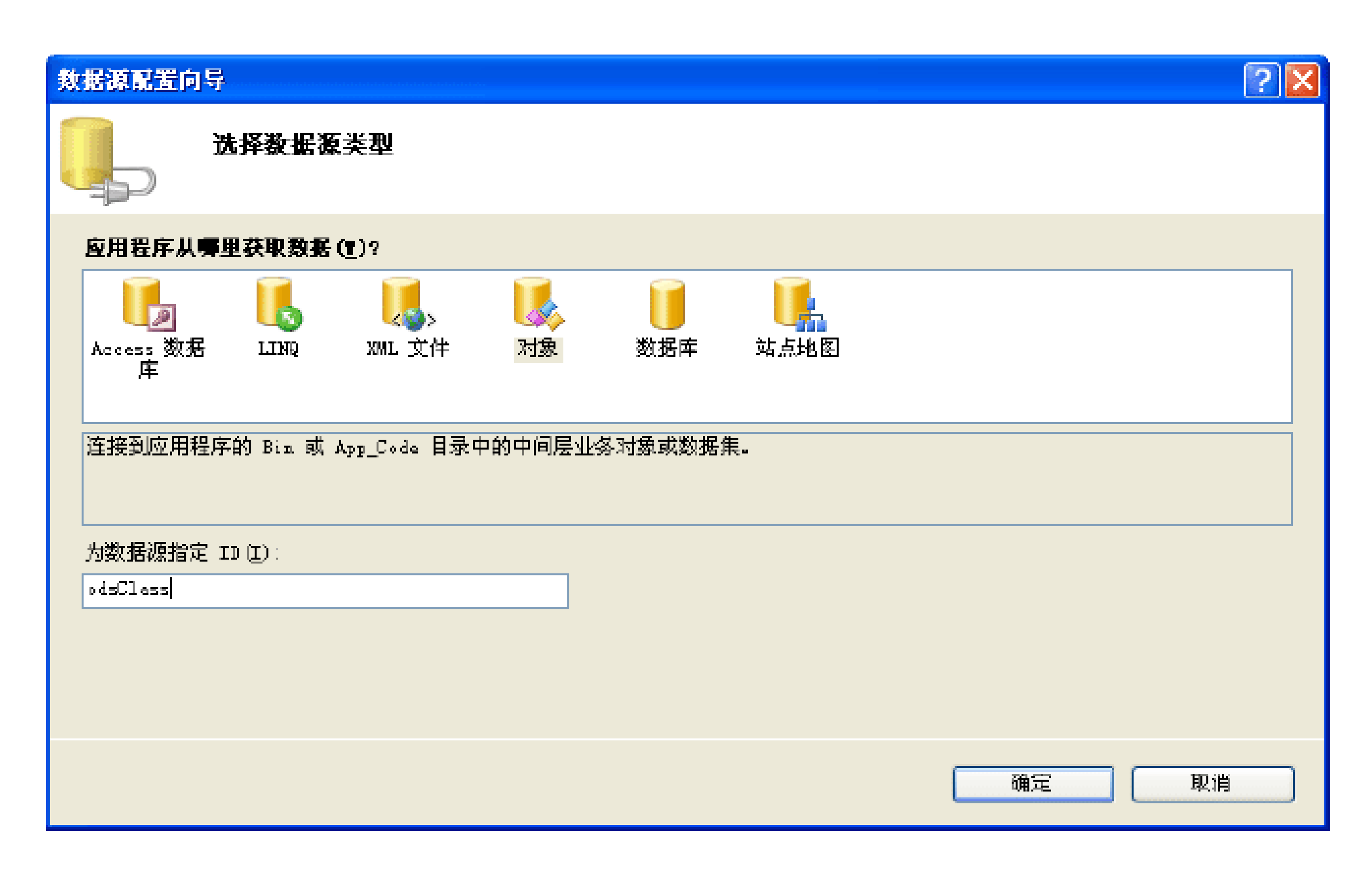


图 6-5 “数据源配置向导”对话框

(4) 在“应用程序从哪里获取数据？”框里选择“对象”，针对下面的“为数据绑定 ID”输入“odsClass”。单击“确定”按钮，打开“配置数据源-odsClass”对话框，如图 6-6 所示。

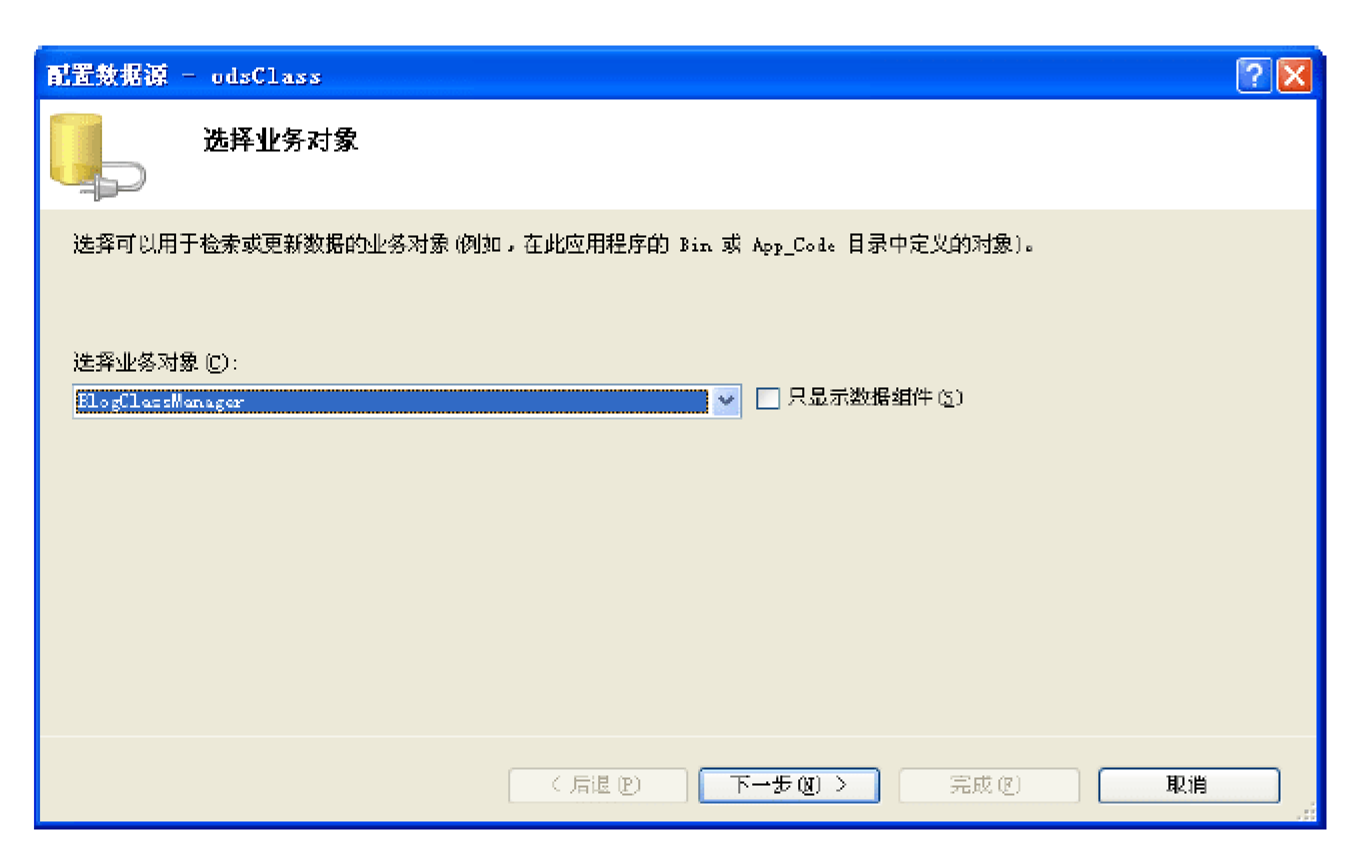


图 6-6 配置数据源-odsClass

(5) 在“选择业务对象”列表框里我们选择要使用的数据访问类 BlogClassManager。单击“下一步”按钮，到“定义数据方法”界面，如图 6-7 所示。

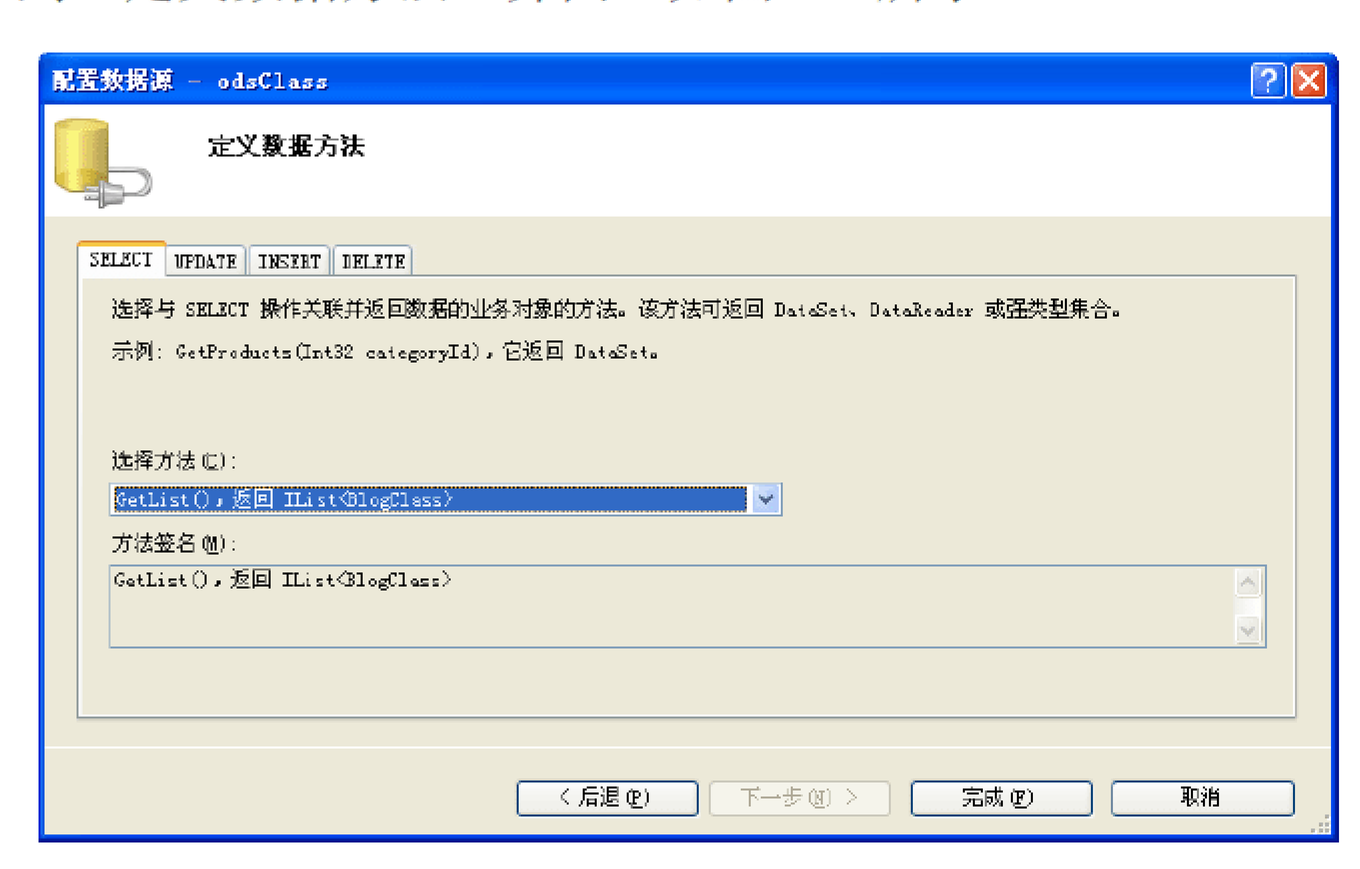


图 6-7 “定义数据方法”界面

(6) 在“选择方法”列表框中选择 GetList 方法。在运行时系统会自动调用该方法获取数据源绑定到数据控件。单击“完成”按钮，结束设置。页面视图如图 6-8 所示。

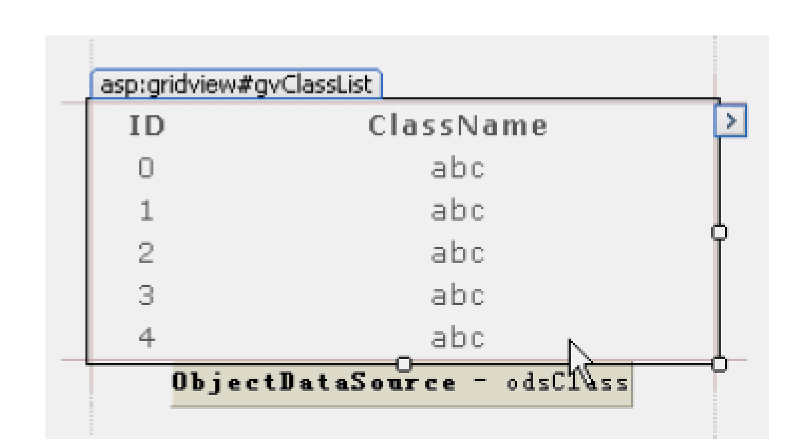


图 6-8　页面视图

可以看到，页面上 GridView 控件的列结构发生了变化，已经是 BlogClass 类的两个属性了，而且 GridView 控件下面多了一个 ObjectDataSource 控件，它是个不可见控件。

(7) 接下来我们来用默认的模板进行美化。单击 GridView 控件右上角的方块，选择“自动套用格式”链接文字。打开“自动套用格式”对话框，如图 6-9 所示。

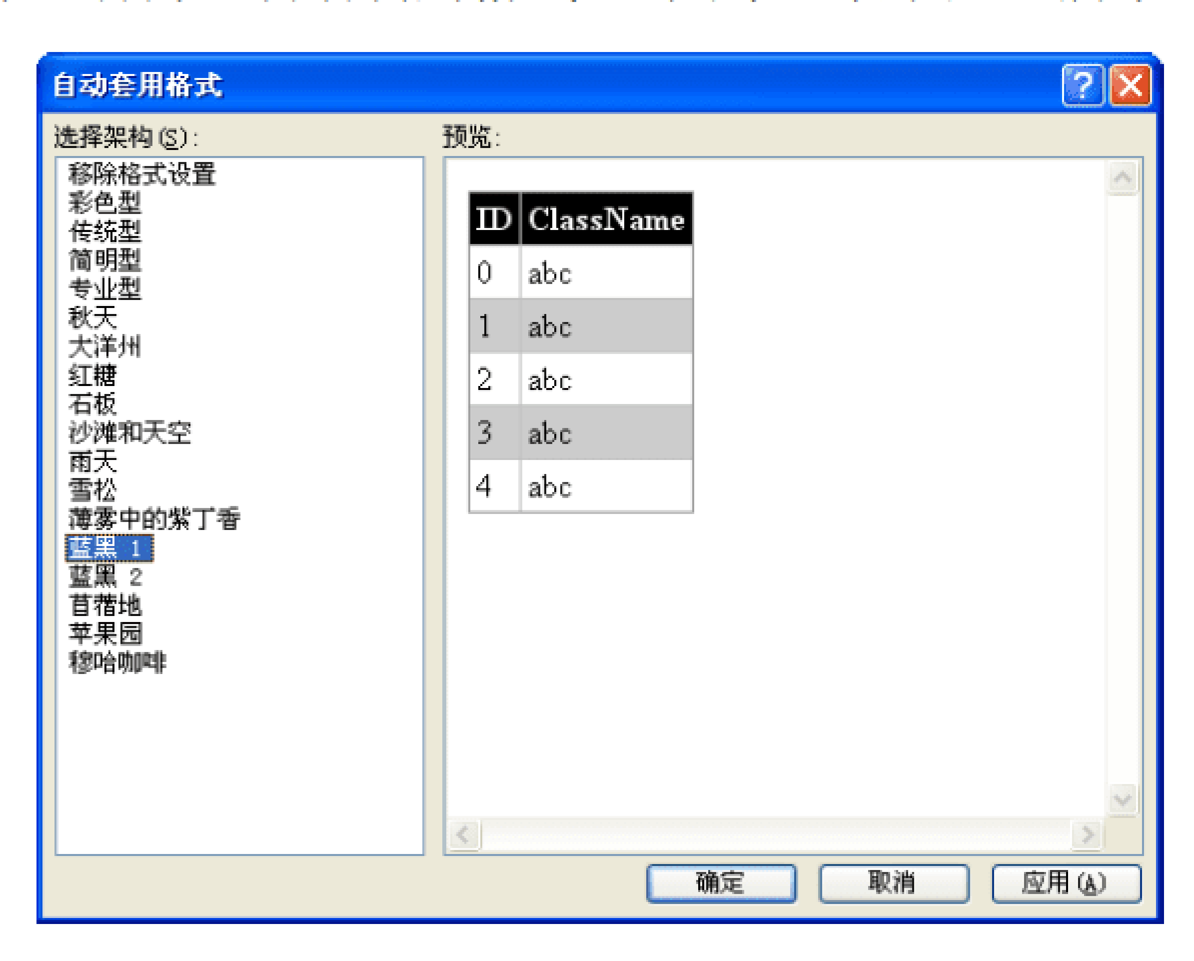


图 6-9　自动套用格式

(8) 在“选择架构”里选择“蓝黑 1”，如图 6-9 所示。然后单击“确定”按钮。Web 窗体视图界面就已绑定好视图效果了，如图 6-10 所示。

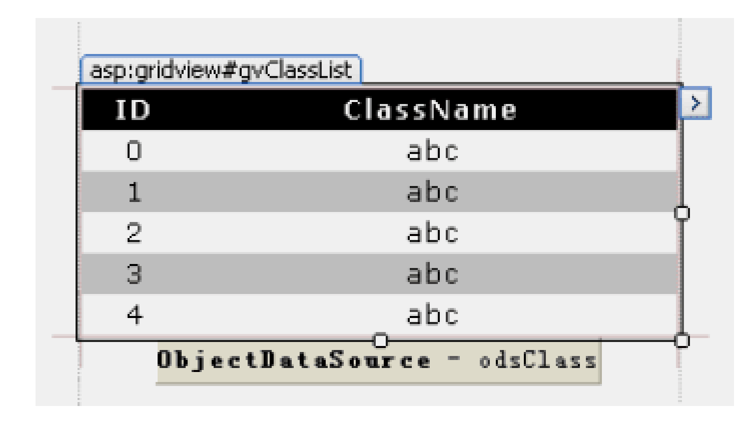


图 6-10　套用格式的GridView

6.3.4　运行结果

运行项目，访问 EditClassList.aspx 页面，如图 6-11 所示。



图 6-11　文章分类列表

6.3.5　实例分析



源码解析：

本实例首先拖动创建了一个 GridView 控件，然后对其创建和绑定 ObjectDataSource 对象。创建 ObjectDataSource 对象需要绑定业务对象和业务对象的方法。然后在“自动套用格式”对话框中对 GridView 控件进行样式设置。

GridView 控件还提供了分页的功能，同样是单击 GridView 控件右上角的小方块“启用分页”即可。

6.4　实现对博文分类的快速编辑

对于一些数据简单的列表，通常我们可直接在列表中涂抹更改数据。这样做方便、快速、直观。比如我们现在的博文分类列表，需要修改的仅仅是名称一列。专门创建一个编辑页面显然有点过于浪费。

在 ASP.NET 里，GridView 控件也提供了直接更新数据源中数据的功能。我们用它就可以直接修改博文分类列表。



视频教学：光盘/videos/6/GridView 编辑.avi　　长度：15 分钟

6.4.1　基础知识——GridView编辑

GridView 控件提供了内置的编辑功能，它能够直接更新其数据源中的数据。该控件为每一个数据行提供了“编辑”、“更新”和“取消”链接按钮，通过它们可以直接修改列表内的数据源。

默认情况下，GridView 控件以只读列表的方式展示数据。单击相应数据行的“编辑”链接按钮时，该数据行处于编辑模式。对应的被编辑字段将显示相应的可操作的编辑控件，用户可以修改数据，然后单击“更新”链接按钮将信息保存到数据源，并回到正常展示状态展示数据，或直接单击“取消”链接按钮回到正常展示状态。

要使用编辑功能，需要首先绑定数据源控件的更新方法。然后启用 GridView 控件的编辑功能即可。

6.4.2　实例描述

在作者的博客里，因为工作太忙，可能很长时间都没有发布一篇新的文章。ASP.NET 分类和 C#分类内容比较接近，所以这里需要把 ASP.NET 分类和 C#分类合并为一个.NET 分类。

具体地，我们就先将 ASP.NET 分类修改为.NET 分类。

下面来实现这个功能。

6.4.3　实例应用

【例 6-4】实现对博文分类的快速编辑。

(1)　运行上节我们使用的项目，打开 EditClassList.aspx 页面。

(2)　在“设计”视图下，单击 GridView 控件右上角的小方块，弹出“GridView 任务”列表框，如图 6-12 所示。

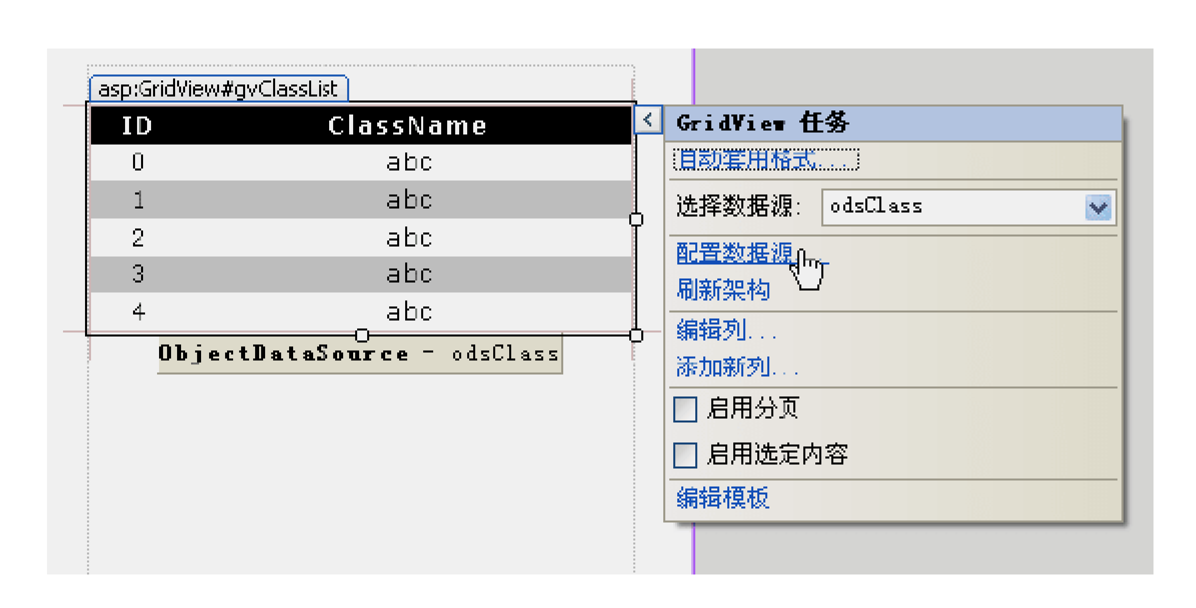


图 6-12　“GridView任务”列表框

(3)　单击“配置数据源”链接按钮，打开“配置数据源”对话框。在选择业务对象界面直接单击“下一步”按钮，来到“定义数据方法”界面。我们需要配置更新方法，在该对话框中单击“UPDATE”选项卡，如图 6-13 所示。



图 6-13　定义数据方法

(4) 在“选择方法”列表框里选择相应的更新方法，这里选择 Update(BlogClass bc)。然后单击“完成”按钮，就配置好了更新方法。

再次单击 GridView 控件右上角的小方块，就可以在“GridView 任务”框中看到“启用编辑”复选框，如图 6-14 所示。

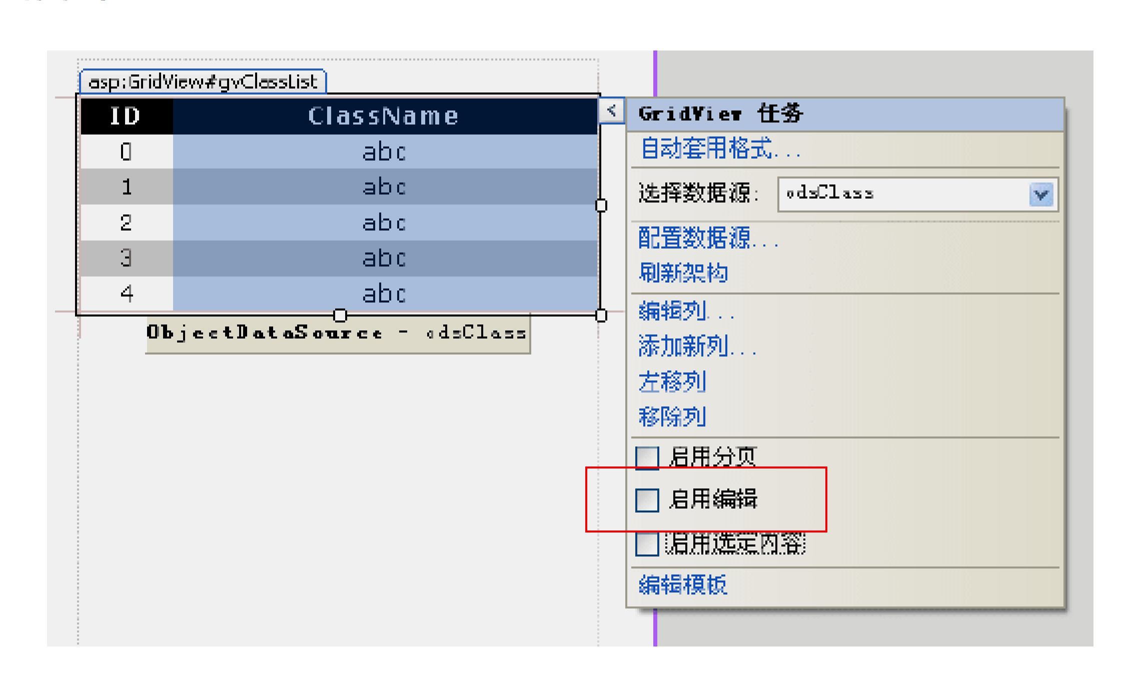


图 6-14　“启用编辑”复选框

(5) 在“GridView 任务”列表框中选中“启用编辑”复选框。GridView 控件会自动增加一个包含“编辑”链接按钮的列，如图 6-15 所示。

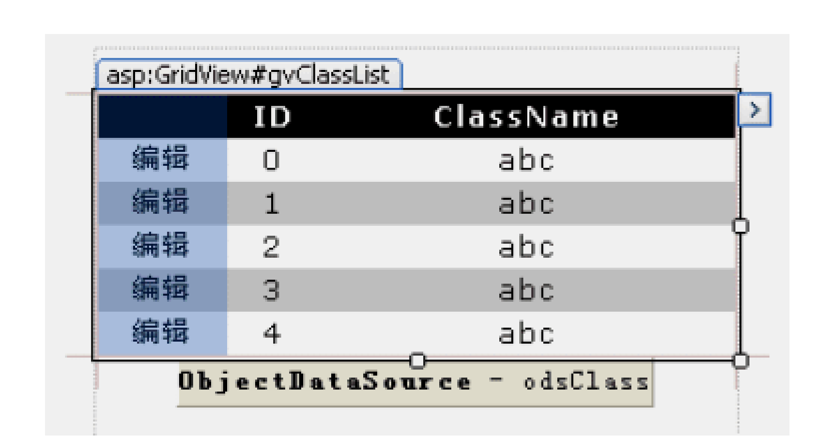


图 6-15　GridView控件的“编辑”列

(6) 在这里，ID 列显示的是数据行的主链，是唯一标识列，所以不需要，也不可以对它进行修改。

单击 GridView 控件右上角的小方块，单击“编辑列”链接按钮，打开“字段”对话框，如图 6-16 所示。

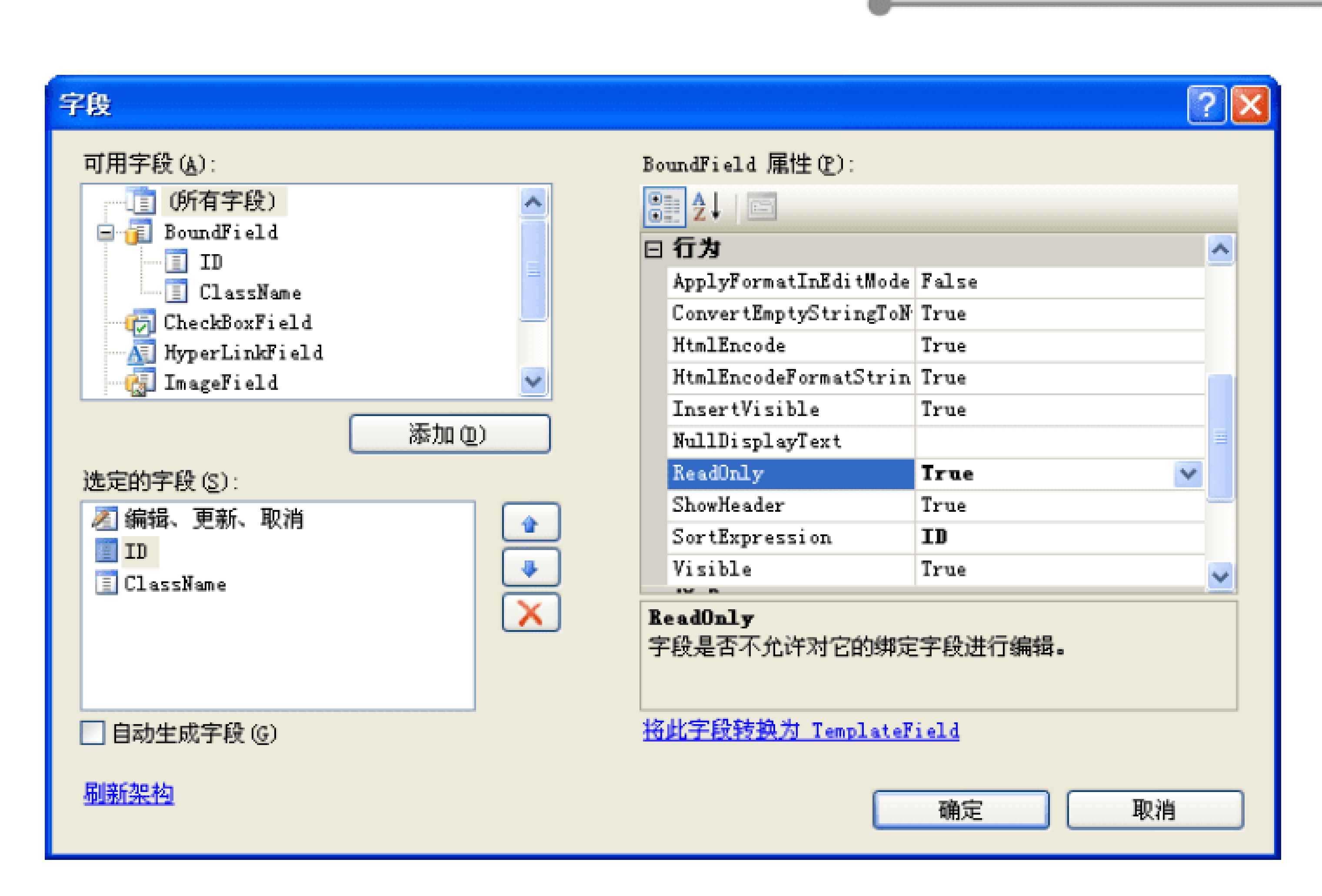


图 6-16　字段

(7) 在“选定的字段”里选择 ID 项，在“BoundField 属性”列表中找到 ReadOnly 项将其值改为 true。

(8) 单击“确定”按钮，即可完成设置。

6.4.4　运行结果

运行项目，访问 EditClassList.aspx 页面，如图 6-17 所示。



图 6-17　分类列表

单击 ClassName 列为 ASP.NET 的行行首的“编辑”链接按钮，该数据行进入编辑状态，如图 6-18 所示。

将值 ASP.NET 修改为.NET，单击“更新”链接按钮，更新并重新绑定 GridView 控件，如图 6-19 所示。

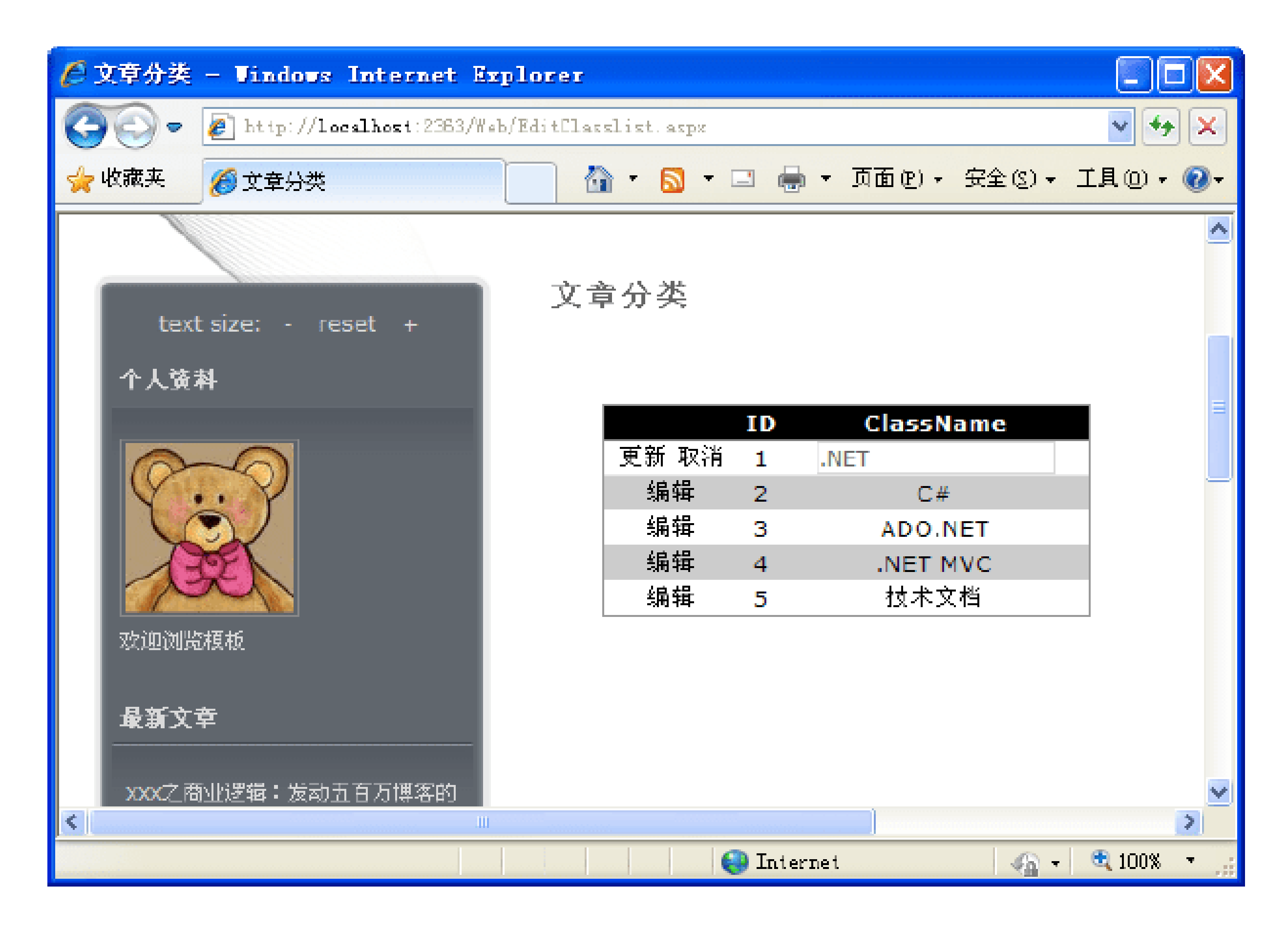


图 6-18　编辑分类

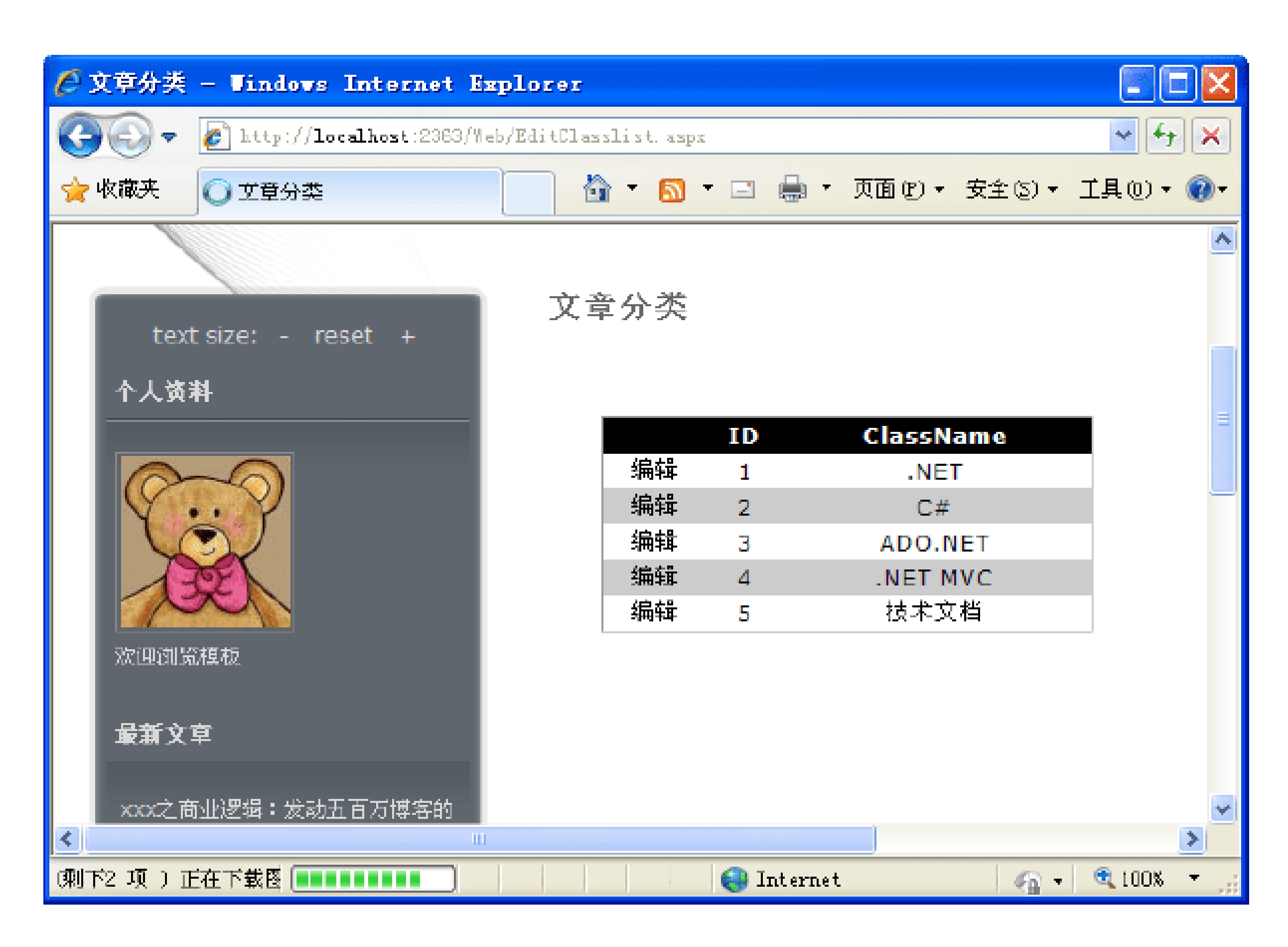


图 6-19　更新成功

6.4.5　实例分析



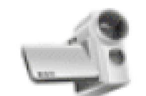
源码解析：

实例首先对 GridView 控件的数据源 odsClass 对象配置了数据更新方法“Update(BlogClass bc)”。然后再启用 GridView 控件的“编辑”功能。为了防止 GridView 控件展示的主键列被修改，最后还在“字段”对话框中设置主键列 ID 为只读列。

6.5　实现对博文分类的删除

在上一章中，我们曾经使用过 GridView 控件实现删除记录功能。不过看那繁杂冗长的代码，实在眼花缭乱。

其实 GridView 控件已经内置了删除功能，我们只要简单地进行设置就可以实现该功能。这一节就使用 GridView 控件内置的东西实现分类删除功能。



视频教学： 光盘/videos/6/GridView 删除.avi　　　　长度：9 分钟

6.5.1　基础知识——GridView删除

GridView 控件提供了内置的删除功能，可以直接删除数据源中的数据。该控件为每个数据行提供了一个“删除”链接按钮。单击该链接按钮即会触发 GridView 控件的删除事件。

当然 GridView 控件的删除功能需要数据源对象 odsClass 的支持。需设置 ObjectDataSource 对象 odsClass 的 DELETE 方法。而 GridView 控件要执行 DELETE 方法，传递的参数的属性名要在 GridView 控件的 DataKeyNames 属性设置一下。

6.5.2　实例描述

在前面的实例中，我们实现了数据的更新，把 ASP.NET 改成了.NET。既如此，C#就显得有点多余了。

在这个实例中我们就编写代码删除它。

6.5.3　实例应用

【例 6-5】 实现对博文分类的删除。

(1)　运行上节我们使用的项目，打开 EditClassList.aspx 页面。

(2)　在“设计”视图下，单击 GridView 控件右上角的小方块，在弹出的“GridView 任务”列表框中选择“配置数据源”，打开“配置数据源”对话框。单击“下一步”按钮，进入“定义数据方法”界面，如图 6-20 所示。



图 6-20　“定义数据方法”界面

(3) 在“选择方法”列表里选择相应的删除业务方法，这里是 Delete(BlogClass bc)，然后单击“完成”按钮，即可完成业务方法的配置。

(4) 在“设计”视图中选中 GridView 控件，在“属性”窗口中找到 DataKeyNames 属性。单击右侧属性值单元格中的“…”按钮，打开“数据字段集合编辑器”对话框，对话框如图 6-21 所示。



图 6-21 “数据字段集合编辑器”对话框

(5) 在“数据字段集合编辑器”对话框中选中“可用数据字段”列表中的项，单击“>”按钮，将“可用数据字段”列表中的所有项都添加到“所选数据字段”列表中。单击“确定”按钮，即可完成对 DataKeyNames 属性的设置，如图 6-22 所示。

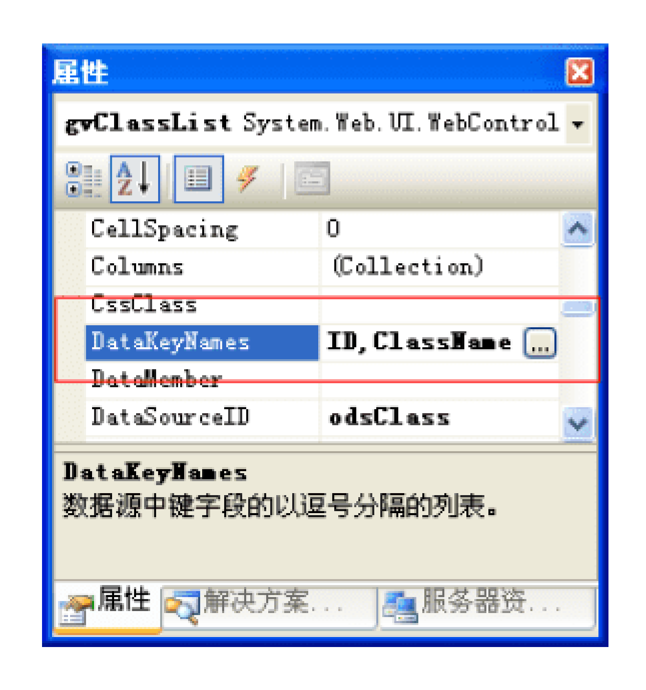


图 6-22 DataKeyNames

(6) 单击 GridView 控件右上角的小方块，可以看到“GridView 任务”列表框里出现了“启用删除”复选框。选中它，就启用了 GridView 控件的删除功能，如图 6-23 所示。

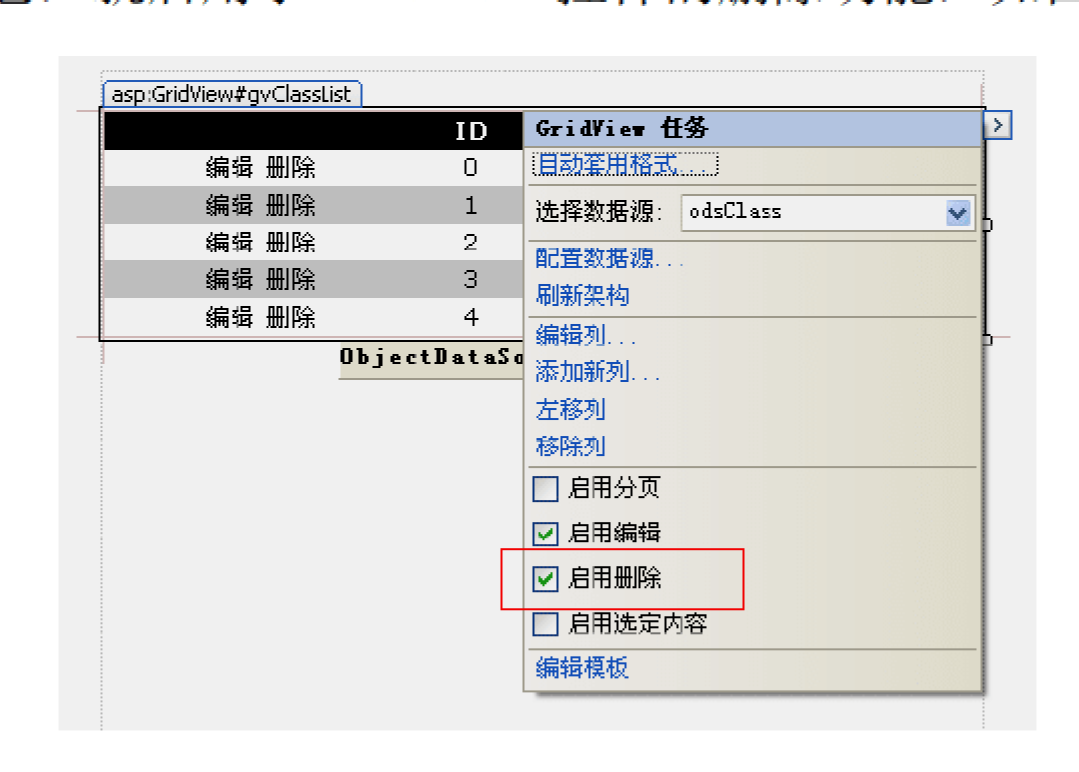


图 6-23 启用删除

6.5.4　运行结果

保存所有修改，运行项目，访问 EditClassList.aspx 页面。

访问结果如图 6-24 所示。

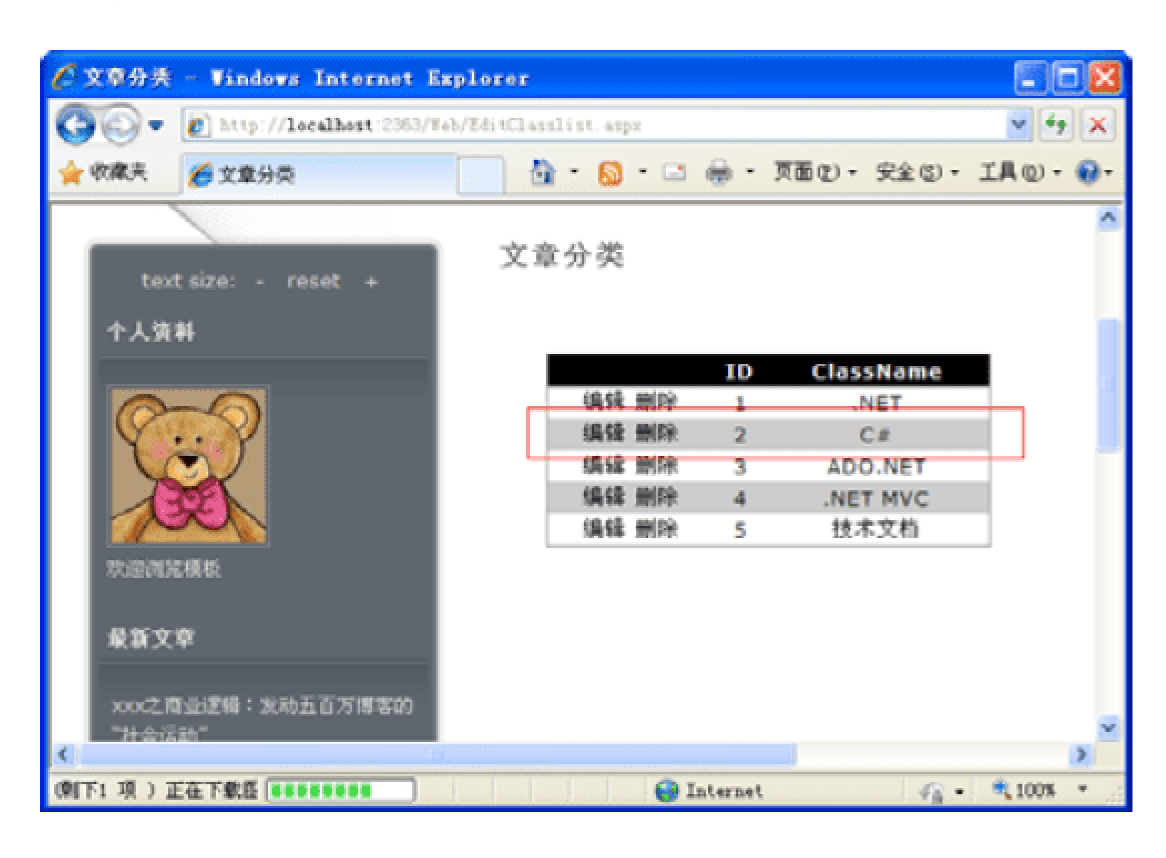


图 6-24　分类列表

我们要删除 ClassName 值为 C#的记录，如图 6-24 红框所示。单击该行行首的“删除”链接按钮，即删除该条记录。删除后的结果如图 6-25 所示。



图 6-25　删除成功

6.5.5　实例分析



源码解析：

实例对 GridView 控件的数据源对象 odsClass 的数据方法进行补充配置。这里设置 DELETE 方法为 Delete(BlogClass bc)。

然后设置 GridView 控件的 DataKeyNames 属性，这里将 BlogClass 的所有属性名称添加进去。

然后在“GridView 任务”列表框里启用 GridView 控件的删除功能即可。

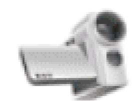
6.6 使用DetailsView控件查看和编辑博客文章信息

上一节我们使用的 GridView 控件，它能一次从数据源中列出大量数据，并通过表格的形式展示出来，每项数据占用一个数据行。这种方式为使用者查看数据列表提供了很大的方便。

但是毕竟数据行宽度有限，在展示较多的项目或是大段的数据内容时显示效果不佳。此时用户可能只想查看一条记录的详细信息。

ASP.NET 提供了一个操作单条记录的具有强大功能的控件——DetailsView。该控件能够以表格的形式，详细地显示每一条记录的每个数据字段的具体内容。

本节将介绍该控件的基本用法，并使用该控件实现博客文章的查看和编辑功能。

 **视频教学：** 光盘/videos/6/DetailsView.avi　　长度：5 分钟

6.6.1 基础知识——DetailsView

DetailsView 控件是一个详细信息视图控件，它能实现展示、更新、插入、删除一条记录的功能。DetailsView 控件需要依赖数据源控件的功能执行更新、插入、删除等任务。

DetailsView 控件使用表格布局，并将记录的每个字段显示于它自己的一行内。DetailsView 控件常用于查看、更新、插入新记录。

DetailsView 控件支持类似于 GridView 的 Fields 集合属性。除了 DetailsView 将每个字段显示一行而 GridView 将所有字段显示在一行之外，功能上与 GridView 控件的 Columns 集合相似。

DetailsView 控件也给每个字段提供了 5 个模板，它们分别是 AlternatingItemTemplate、EditItemTemplate、HeaderTemplate、InsertItemTemplate、ItemTemplate。

DetailsView 控件本身也有 4 个模板：PagerTemplate、EmptyDataTemplate、FooterTemplate、HeaderTemplate。

DetailsView 控件对每一个模板提供了一个样式属性，我们可以使用它们为对应的模板设置属性。

6.6.2 实例描述

在我的博客系统中，博客文章的操作管理功能显得尤为重要。

刚才说到 DetailsView 控件具有增、删、改、查等诸多强大的功能。这里就使用 DetailsView 控件来实现博客文章的修改功能。

6.6.3 实例应用

【例 6-6】使用 DetailsView 控件查看和编辑博客文章信息。

(1) 运行我们的 Blog 项目，创建一个新的 Web 窗体，命名为 EditBlog.aspx。

(2)　在页面内拖入一个 DetailsView 控件，设置 ID 属性为 dvBlog，Width 属性为 90%，如图 6-26 所示。

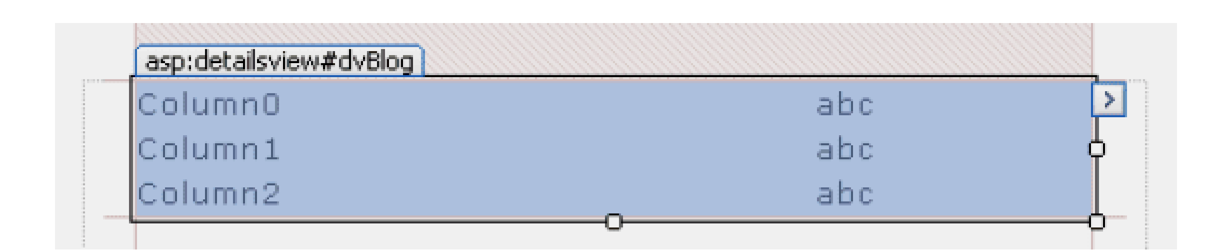


图 6-26　DetailsView

(3)　单击控件右上角的小方块，打开 Details 任务框。在“选择数据源”列表框中选择“新建数据源”选项。

(4)　在“选择数据源类型”界面，我们选择“对象”数据源，并命名为 odsBlog，单击“确定”按钮。

(5)　在“选择业务对象”界面中，选择我们已经创建好的业务对象 BlogManager，单击“下一步”按钮。

(6)　在“选择数据方法”界面，在 SELECT 选项卡里选择一个查询方法，这里我们使用 GetById(Int32 id)方法，在 UPDATE 选项卡里选择 Update(Blog b)方法。然后继续单击“下一步”按钮。

(7)　我们要对 SELECT 方法设置参数，在“参数”列表中选择参数 id，在“参数源”列表中选择选项 QueryString，表示该参数从请求字符串中获取值。在 QueryStringField 文本框中输入 id，这里对应 Web 请求的参数名，如图 6-27 所示。

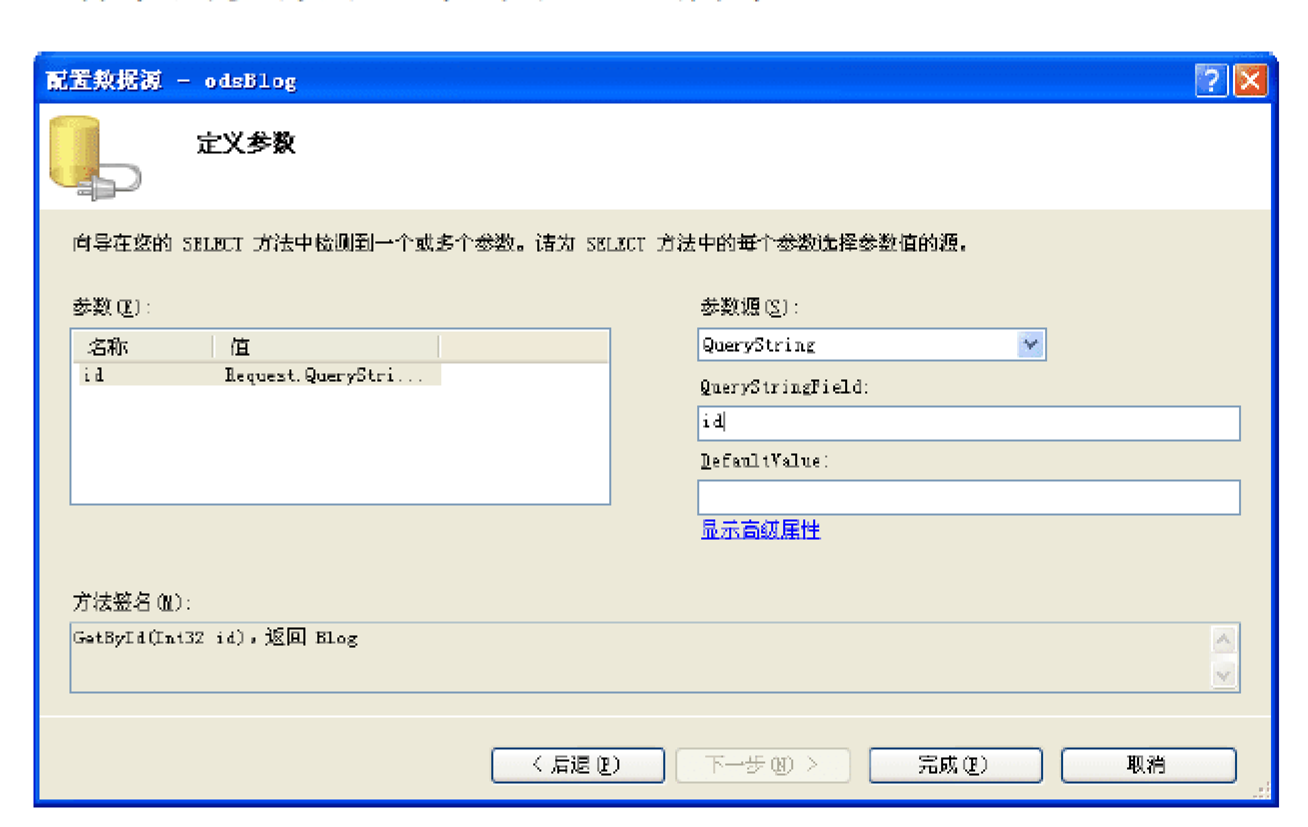


图 6-27　定义参数

(8)　单击“确定”按钮完成对数据源的设置，如图 6-28 所示。

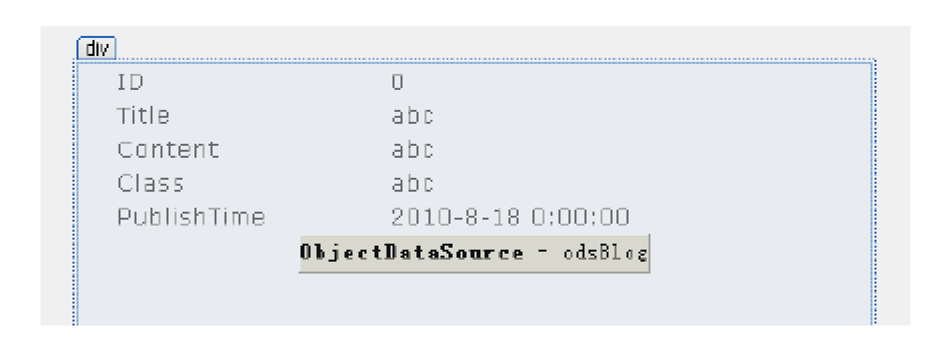


图 6-28　绑定数据源

(9)　在“Details 任务框”中选中“启用编辑”复选框，即可启用 Details 的编辑功能。

(10) 在“Details 任务框”中单击“自动套用格式”，在打开的“自动套用格式”对话框中

为 Details 控件设置样式。这里选择“专业型”，单击“确定”按钮。结果如图 6-29 所示。

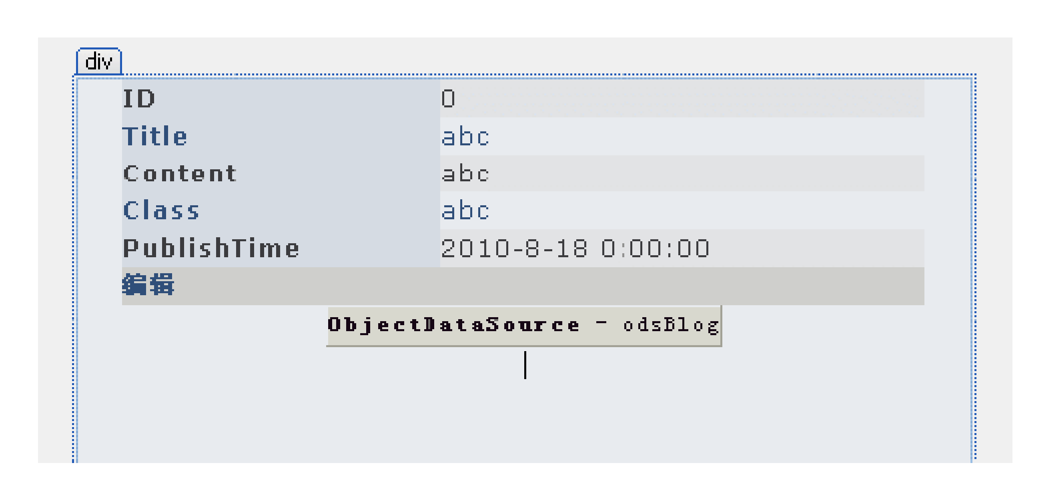


图 6-29　启用编辑并设置样式

(11) 在这里，ID 字段是主键，不能修改，PublishTime 是发布时间，不可以修改，所以我们需要将这两个字段设为“只读”。在“Details 任务框”中单击“编辑字段”，打开“字段”对话框。在“选定的字段”列表中依次选中这两个字段，并在“BoundField 属性”列表中设置 ReadOnly 属性值为 true。

(12) 对于文本类型，数据系统默认使用文本框来编辑数据，而 Content 属性内容较多，所以我们需要使用多行文本域来编辑它，这里需要编辑该字段。

在“字段”对话框中“选定的字段”列表中选中该 Content 字段，单击“将此字段转换为 TemplateField”链接按钮，如图 6-30 所示。

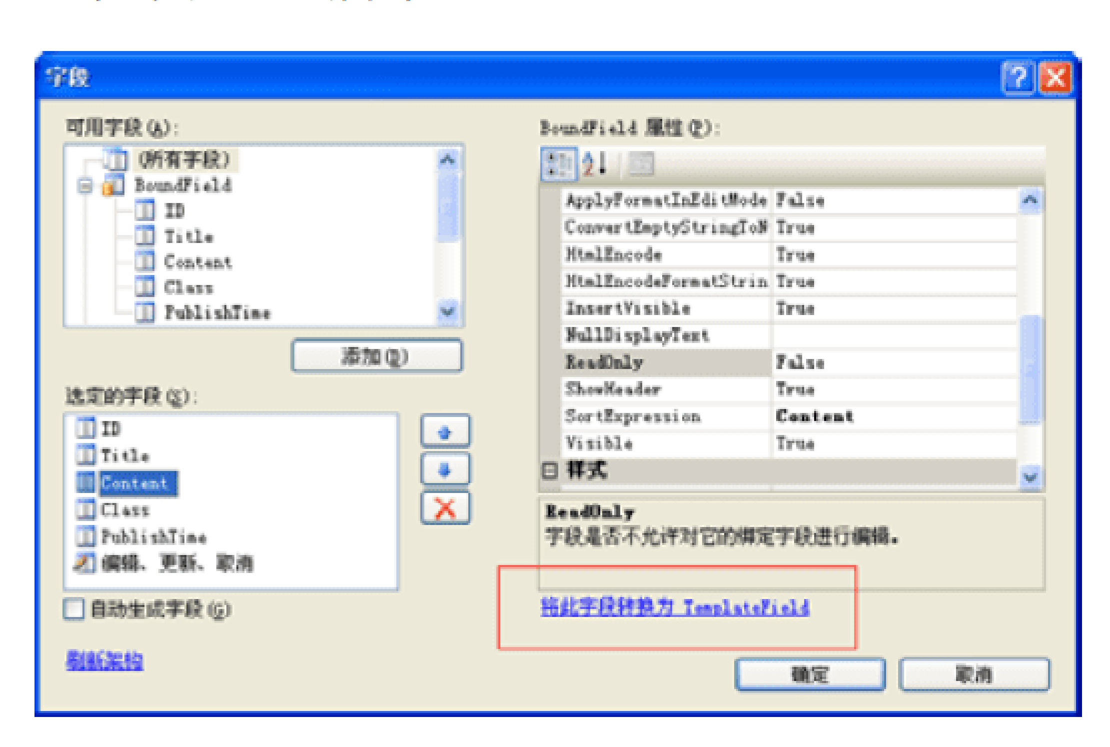


图 6-30　“字段”对话框

(13) 单击“确定”按钮回到 Web 窗体。

(14) 重新打开“Details 任务”框，单击“编辑模板”链接文本，进入“模板编辑模式”。

(15) 在“Details 任务”框中的“显示”列表框中选择 Content 下面的 EditItemTemplate 项，进入“编辑模式”的模板编辑状态。

(16) 修改域内文本框 TextBox1 的 TextMode 属性为 MultiLine，Width 属性为 300，Height 属性为 60，效果如图 6-31 所示。

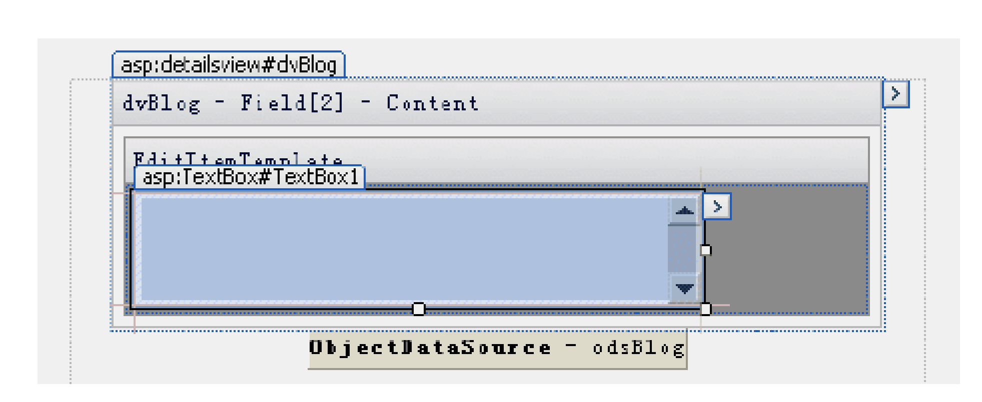


图 6-31　编辑模式

(17) 在“Details 任务”框里单击“结束模板编辑”。保存文件。

(18) 当然，我们还要在博文列表页面进入编辑状态，打开 Default.aspx 页面，在模板里“发表时间”项目后面加入如下一行代码：

```
<span><a href='<%# "EditBlog.aspx?id=" + Eval("Id") %>'>编辑</a></span>
```

保存，完成该实例。

6.6.4　运行结果

运行项目。

先来访问 Default.aspx 页面，打开博客文章列表，如图 6-32 所示。

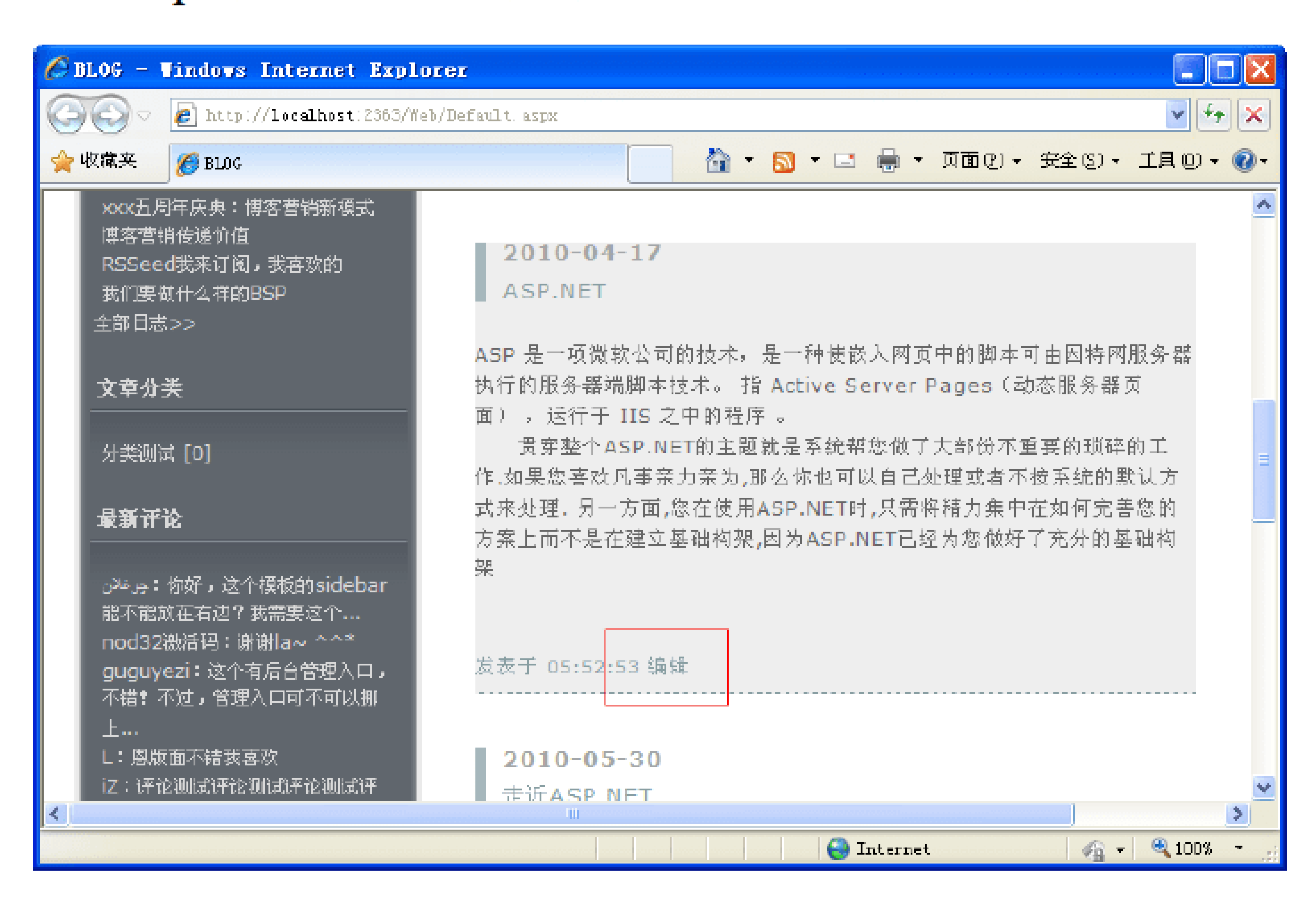


图 6-32　博客文章列表

我们单击图 6-32 中红色线框标记的“编辑”链接，打开编辑页面，如图 6-33 所示。



图 6-33　编辑页面

再单击展示表格下面的“编辑”链接，进入编辑状态，如图 6-34 所示。

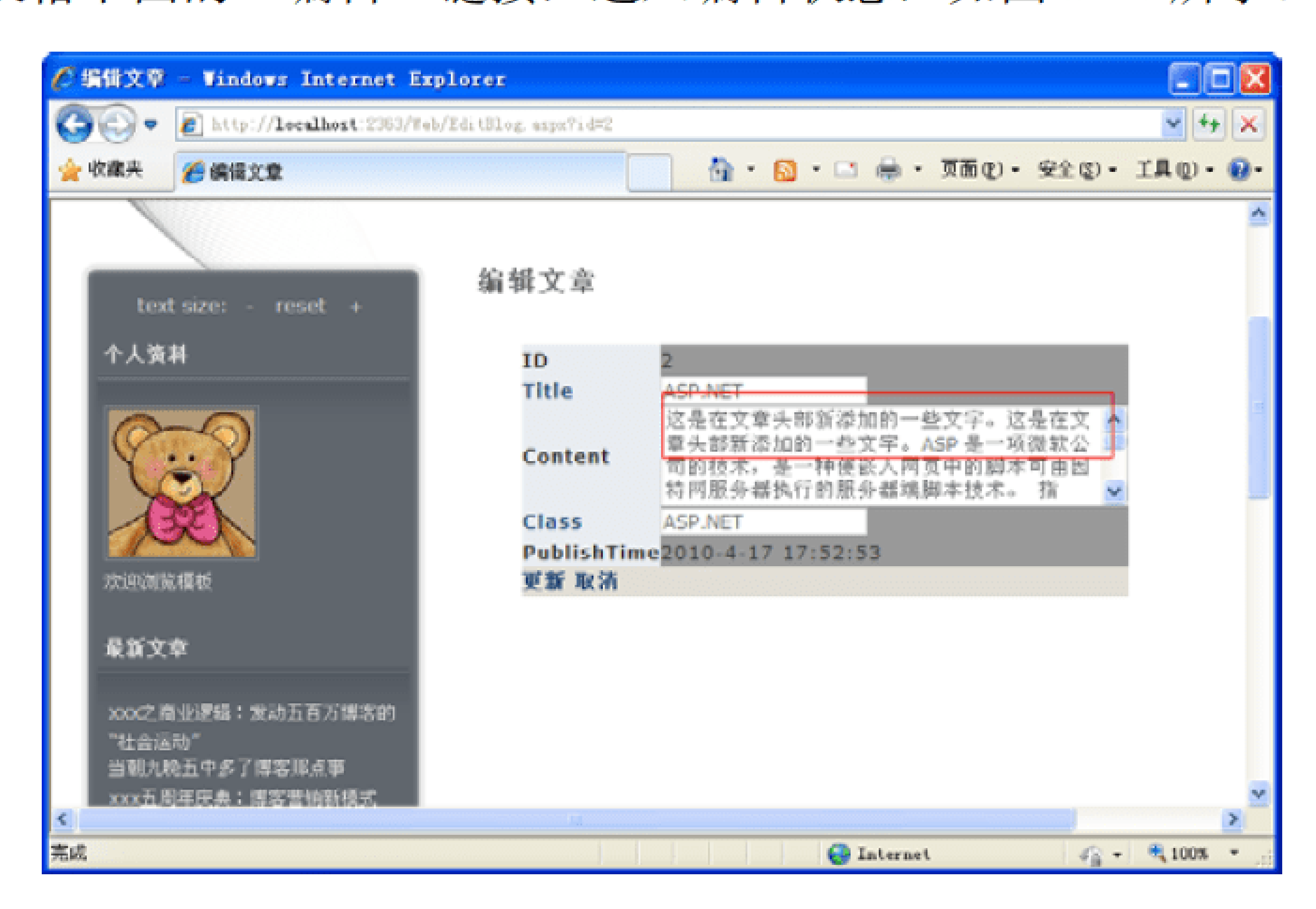


图 6-34　编辑状态

在文章内容中添加一些文字，如图 6-34 中红线框内所示。

单击“更新”链接，保存数据，再来访问 Default.aspx，结果如图 6-35 所示。



图 6-35　文章列表

可以看到我们添加的内容已经成功地保存到数据库中。

6.6.5　实例分析



源码解析：

本实例先创建一个文章编辑页面，接着创建一个 DetailsView 控件，并对该数据控件创建和绑定数据源，然后修改 ID 字段和 PublishTime 字段为只读属性，然后将 Content 字段转换为 TemplateField，并编辑其 EditItemTemplate 模板为一个多行文本域控件，最后添加博客文章列表页到本页的超级链接。

Web程序员成功之路

6.7 自定义博客文章展示布局

上一节我们使用 DetailsView 控件对博客文章列表进行查看和修改。

使用 Details 确实方便、简单。但是因为其固定使用表格布局，很难自由地调整页面效果。不过 ASP.NET 还提供了一个类似于 DetailsView 控件的控件——FormView。它完全自定义布局，和 Repeater 控件一样，不在用户界面生成任何多余的代码。

 视频教学：光盘/videos/6/FormView.avi　　长度：6 分钟

6.7.1 基础知识——FormView

FormView 控件和 DetailsView 控件都能够实现一次显示、编辑、插入、删除一条记录的功能。它们十分类似，但还不完全相同。它们的区别在于：

- FormViewr 控件使用模板来自定义指定项的布局。
- DetailsView 控件使用表格布局，并且记录的每个字段都各自显示为表格的一行。

FormView 提供了 7 个模板，分别是 EditItemTemplate、EmptyDataTemplate、FooterTemplate、HeaderTemplate、InsertItemTemplate、ItemTemplate、PagerTemplate。具体用法和功能前在前面几个控件中都已经讲过，这里大同小异，不再详细说明。

6.7.2 实例描述

博客文章列表一般来说只显示文章的部分内容。

因为往往有一些比较长的文章，如果在列表中显示全文的话，列表页面会非常非常的长。给页面浏览者很不好的感觉。所以我们需要把博客文章的详细内容展示到一个单独的页面。

前面我们用过一个展示单条记录的控件 GridView，由于生成的是表格布局，所以页面效果很难控制。而现在越来越炫的页面效果要求，显然已经不适合再使用古板的 DetailsView 了。

所以在这一节我们使用 FormView 控件来控制博客文章的展示。

6.7.3 实例应用

【例 6-7】自定义博客文章展示布局。

(1) 运行我们的 Blog 项目，创建一个新的 Web 窗体，命名为 BlogShow.aspx。

(2) 在页面拖入一个 FormView 控件，设置其 ID 属性为 fvBlog。

(3) 在“设计”视图下单击 FormView 控件右上角的小方块，在打开的 FormView 任务框里的“选择数据源”列表框中选择“新建数据源”选项。

(4) 在“数据源配置向导”中选择数据源类型为“对象”，并指定其 ID 为 odsBlog。

(5) 单击“确定”按钮，选择业务对象 BlogManager，单击“下一步”按钮，在 SELECT 选项卡中选择 GetById(Int32 id)，单击“下一步”按钮，选中参数 id，在参数源里选择 QueryString 选项，在 QueryStringField 文本框里输入 id，单击“完成”按钮。结果如图 6-36 所示。



图 6-36 FormView绑定数据源

(6) 我们切换到“源”视图，可以发现系统自动为我们创建了展示、添加、修改等 3 个模板。这里用不到添加和修改功能，所以我们删掉 InsertItemTemplate 和 EditItemTemplate 两个模板，只留一个 ItemTemplate。

(7) 编辑 ItemTemplate 里的布局代码，如下所示：

```
<asp:FormView ID="fvBlog" runat="server" DataSourceID="odsBlog">
    <ItemTemplate>
        <div class="postHeader">
            <H3><%# Eval("PublishTime","{0:yyyy-MM-dd hh:mm:ss}") %></H3>
            <H2><A href="#"><%# Eval("Title") %></A></H2>
        </div><div class="clear"></div>
        <div class="postBody">
            <P><%# Eval("Content") %></P>
            <div class="clear"></div>
        </div>
    </ItemTemplate>
</asp:FormView>
```

至此已经完成 BlogShow.aspx 页面的所有功能，接下来我们修改列表页面代码，让用户在单击列表中文章标题的时候查看相应文章的内容。

(8) 打开 Default.aspx，修改 Repeater 两个模板里绑定标题的代码如下：

```
<H2>
<A href='<%# "BlogShow.aspx?id=" + Eval("Id") %>'><%# Eval("Title") %>
</A>
</H2>
```

(9) 保存所有文件，就完成了所有的功能。

6.7.4 运行结果

运行项目。

访问 Default.aspx 页面，单击某一篇文章标题，打开详细展示页面，如图 6-37 所示。

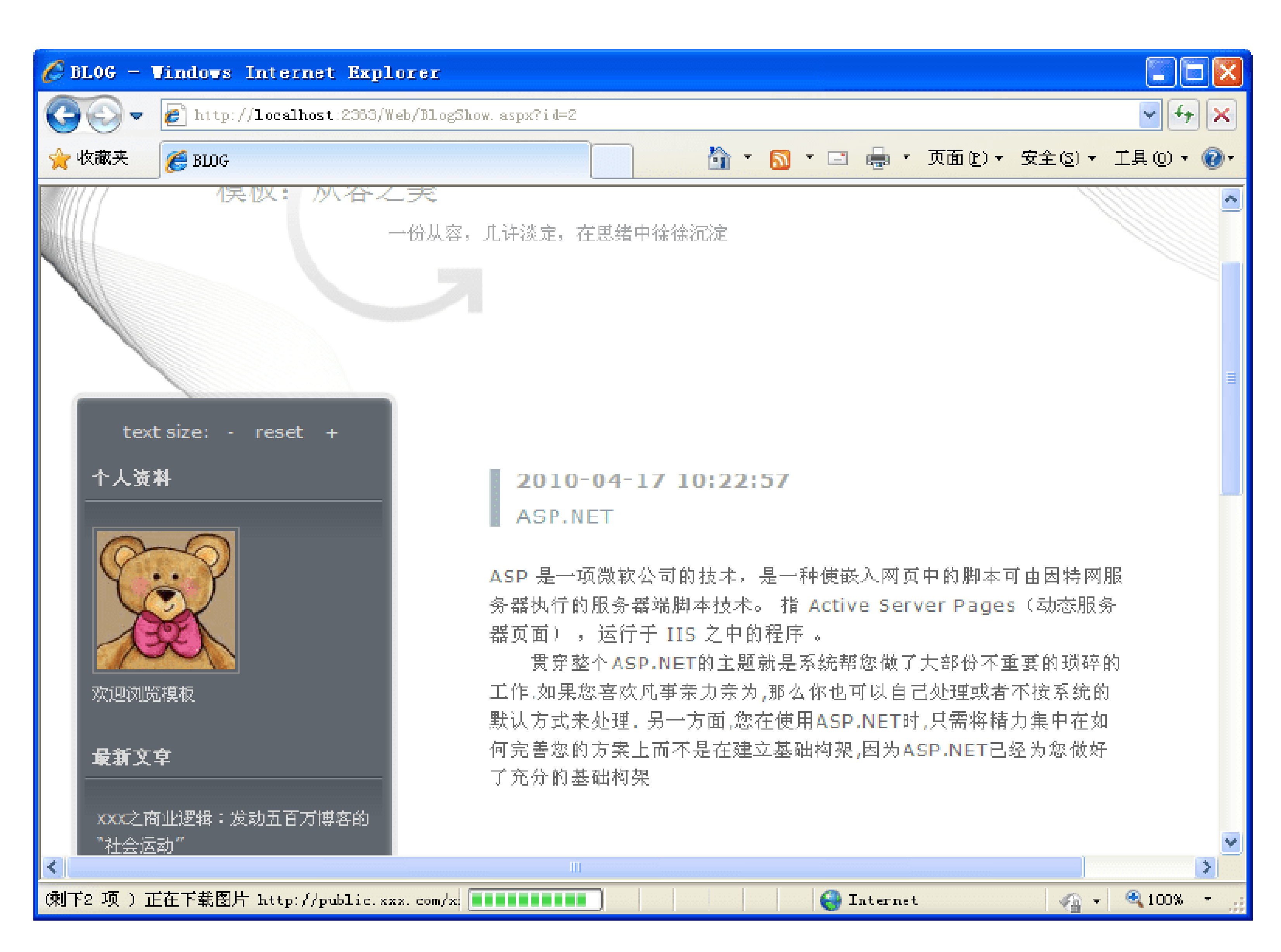


图 6-37　文章展示

6.7.5　实例分析


源码解析：

本实例使用 FormView 控件展示博客文章内容。实现步骤非常简单，先是创建和绑定数据源，然后修改 FormView 控件的布局模板，最后在博客文章列表页面设置超链接指向该页面。

其中在修改 FormView 控件的布局模板的时候，该模板设置方式和注意事项与 Repeater 控件的模板设置一模一样。

6.8　常见问题解答

6.8.1　如何取得GridView中控件的值



如何取得 GridView 中控件的值？

网络课堂：http://bbs.itzcn.com/thread-5065-1-1.html

在 GridView 中，有一列是 TextBox 控件，绑定了某个值，如何提取这些 TextBox 的值呢？

【解决办法】：试试下面的这种做法：

```
TextBox txt = (TextBox)this.GridView.Rows[0].Columns[0].FindControl("txtID");
string str = txt.Text;
```

在上述语句中，首先查找 GridView 控件中的第一行第一列中 id 为 txtID 的控件，并将该

控件转换为 TextBox 类型，然后赋值给 txt 控件。最后获取 txt 控件中的文本值并赋值给 str。

其中，FindControl()方法用于查找某个控件容器中的其他控件。

6.8.2 DataList控件问题



DataList 控件问题。

网络课堂：http://bbs.itzcn.com/thread-5094-1-1.html

DataList 控件中有一个 RadioButton，如何控制 DataList 数据绑定后 RadioButton 只能选择一个？

【解决办法】：试试下面的代码，作者可是冥思苦想了整整一上午啊：

```
DataGrid.Items[0].Cells[0].RowSpan = DataGrid.Items.Count;
for (int i=1; i<DataGrid.Items.Count; ++i)
{
    DataGrid.Items[i].Cells[0].Visible = false;   //从第二列开始隐藏第一个单元格
}
for (int i=0; i<DataGrid.Items.Count; ++i)
{
    ListItem ss = new ListItem(" ", "1");
    ((RadioButtonList)DataGrid.Items[0].Cells[0].Controls[1]).Items.Add(ss);
     //将第一列第一个单元格里的 RadioButtonList 按照 DataGrid 的总列数进行列添加
}
```

6.8.3 FormView中的控件问题



FormView 中的控件问题。

网络课堂：http://bbs.itzcn.com/thread-3443-1-2.html

FormView 的插入模式下，有一个发布日期的 TextBox 控件，绑定的是数据库中的日期字段，现在想实现一打开这个页面，该 TextBox 控件就显示当天的日期，加了如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    ((TextBox)FormView1.FindControl("txtCreatetime")).Text =
      DateTime.Now.ToLongDateString();
}
```

第一次打开页面可以自动显示日期，但是，当点击插入后，TextBox 控件就会变为空的，该怎样实现这样的效果呢？

【解决办法】：在 DataBound 事件中判断当前的模式是否为 Insert：

```
protected void FormView1_DataBound(object sender, EventArgs e)
{
    if(FormView1.CurrentMode.ToString() == "Insert")
```

Web 程序员成功之路

```
        ((TextBox)FormView1.FindControl("txtCreatetime")).Text =
            DateTime.Now.ToLongDateString();
}
```

首先你应该先判断当前的模式是否为插入模式，如果是，则将当前日期转换为长日期形式，然后在 FormView1 控件中查找 id 为 txtCreatetime 的控件并将该控件转换为 TextBox 类型。最后将转换后的日期赋值给 id 为 txtCreatetime 的控件。

6.9　习　　题

一、填空题

(1) ASP.NET 提供了__________种类型的数据源。

(2) 我们为页面数据展示控件绑定已存在的数据源控件的时候需要使用该数据展示控件的__________属性。

(3) 在 ASP.NET 中，将字段设置为__________时，可以对该列随意地编辑。

(4) 在 ASP.NET 的数据展示控件中，对字段数据进行绑定有两个方法：Eval 和________。

(5) 在 ASP.NET 提供的 Repeater、DataList、GridView 三个控件中，使用起来灵活度最高的是__________控件。

(6) 在 DetailsView 和 FormView 两个控件中，__________是固定由表格布局的。

二、选择题

(1) 在 ASP.NET 中，已知在页面上为一个 DataList 控件设置了 ObjectDataSource 数据源，对象类型为 Customer。要在页面上绑定 Age 字段，下面绑定表达式正确的是__________。

A. <%# Eval(Customer.Age) %>　　B. <%# Customer.Age %>

C. <%# Eval("Age") %>　　D. <%# "Customer.Age" %>

(2) Repeater 与 GridView 控件相比最重要的区别是__________。

A. 能够存储数据　　B. 外观比较美观

C. 显示的布局几乎不受限制　　D. 数据量受一定的限制

(3) GridView 控件中，可以在一行内显示出__________条记录。

A. 1　　B. 2　　C. 3　　D. 多

(4) DataList 控件 RepeatDirection 属性设置为 Vertical，RepeatColumns 属性设置为 4。当数据源中只有 3 条记录的时候，页面将显示__________列数据。

A. 4　　B. 3　　C. 2　　D. 1

(5) 下列控件中，__________控件具有交替项模板。(多选)

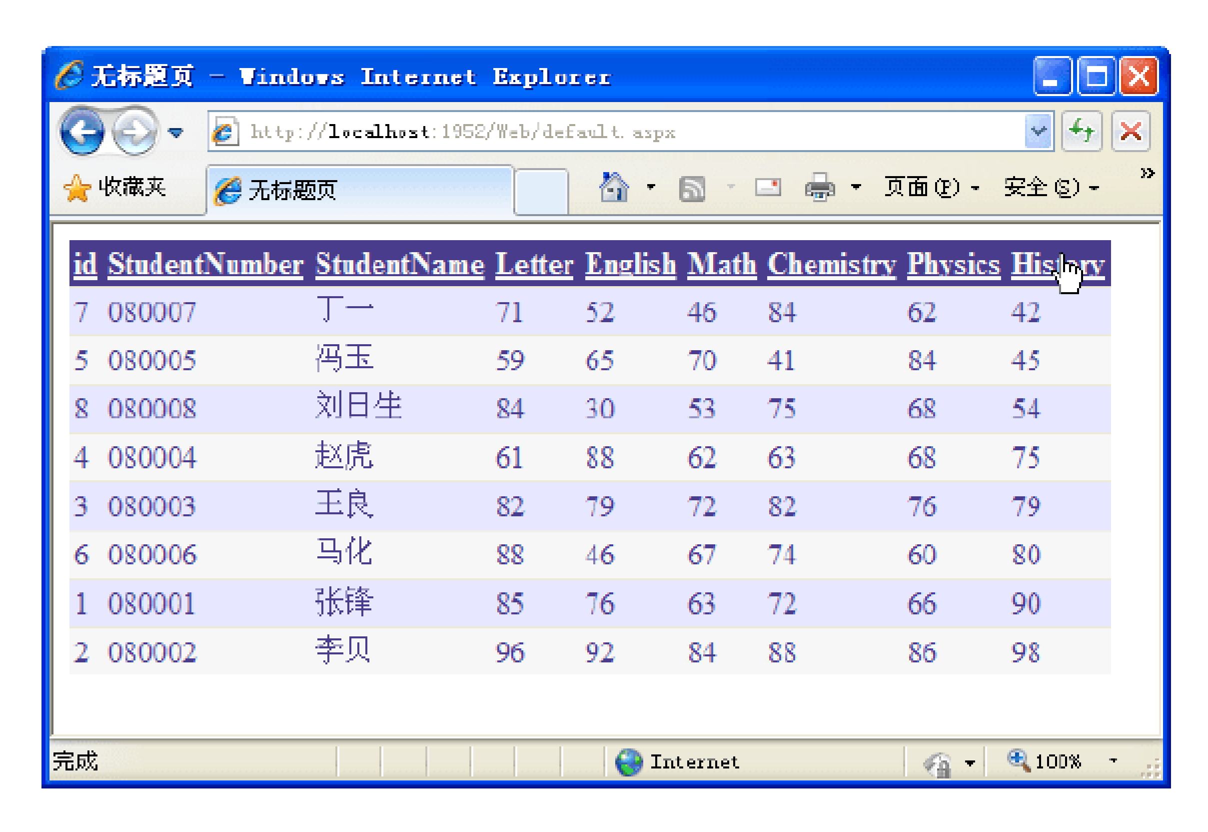
A. Repeater　　B. DetailsView　　C. FormView　　D. DataList

三、上机练习

上机练习：学生成绩查询。

在学生成绩管理系统中，老师们经常要对学生的成绩进行查询，并按相应的字段进行排序显示。

该练习我们就使用 GridView 控件提供的自动排序功能对学生成绩进行查看。如图 6-38 所示的列表是按历史成绩(History)排序的结果。



id	StudentNumber	StudentName	Letter	English	Math	Chemistry	Physics	History
7	080007	丁一	71	52	46	84	62	42
5	080005	冯玉	59	65	70	41	84	45
8	080008	刘日生	84	30	53	75	68	54
4	080004	赵虎	61	88	62	63	68	75
3	080003	王良	82	79	72	82	76	79
6	080006	马化	88	46	67	74	60	80
1	080001	张锋	85	76	63	72	66	90
2	080002	李贝	96	92	84	88	86	98

图 6-38　学生成绩列表



第 7 章　与 Web 2.0 的故事

内容摘要：

在商界有这样一句话“得用户者得天下”，意思就是说，如果要想在市场上发展得好，就必须赢得更多用户(客户)的认可。

纵观 Web 的发展历史也是一样的，在最初的 Web 1.0 时代强调以数据为中心，为用户提供一个广泛的信息浏览平台。Web 1.0 的发展出现了向综合门户合流现象，尤其是对于新闻信息，例如早期的新浪、搜狐和网易等门户网站。

在 Web 1.0 时，信息的产生模式是单向的，即用户通过浏览器获取信息。而 Web 2.0 的出现则是以用户为中心，强调的是用户的参与和价值的产生。

也就是说 Web 2.0 网站的内容通常是用户发布的，使得用户既是网站内容的浏览者也是网站内容的制造者，这也就意味着 Web 2.0 网站为用户提供了更多参与的机会。例如，此时的博客和 wiki 就是基于用户创造内容的代表。

另外，Web 2.0 更加注重用户交互性。主要表现为引用了一个关键组件——Ajax，它使我们正在步入一个史无前例的用户界面创新阶段，因为可以像浏览本地基于 PC 的应用程序一样使用丰富的网络程序了。

在本章中，我们将以 ASP.NET 技术为切入点，详细讲解它如何应对实现 Web 2.0 网站，如基于客户端的验证表单、异步请求时间、无刷新注册验证以及仿 Baidu 的自动完成等。

学习目标：

- 理解 Ajax 的运行机制
- 熟悉 Ajax 客户端的验证方法
- 掌握 Ajax 中客户端与服务器的数据传输方式
- 掌握 GET 和 POST 的数据通信
- 掌握 ASP.NET Ajax 部署方法
- 掌握 UpdatePanel 控件的异步更新
- 熟悉 Timer 控件的定时调用
- 熟悉 ASP.NET Ajax 控件的实现

7.1 创建跨浏览器的 XMLHttpRequest 对象

XMLHttpRequest 对象是当今所有 Ajax 和 Web 2.0 应用程序的技术基础。尽管软件经销商和开源社团现在都在提供各种 Ajax 框架以进一步简化 XMLHttpRequest 对象的使用；但是，我们仍然很有必要理解这个对象的详细工作机制。首先来介绍这个对象是如何创建的。

 视频教学：光盘/videos/7/Ajax.avi　　长度：6 分钟

7.1.1 基础知识——Ajax 核心知识

1. Ajax 工作原理

Ajax 全称是异步 JavaScript 和 XML(Asynchronous JavaScript And XML)，是一种在 2005 年由 Google 推广开来的编程模式。Ajax 不是一种新的编程语言，而是使用现有标准的新方法，可谓是新瓶装旧酒。



在 Ajax 中使用的 Web 标准已被定义良好，并被所有的主流浏览器支持，而且这些标准已被大多数开发者使用多年，包括 JavaScript、XML、HTML 和 CSS。

但是使用 Ajax 可以不必刷新整个页面，只是对页面的局部进行更新，而且还可以节省网络带宽、提高网页加载速度，从而缩短用户等待时间，改善用户操作体验。基于 Ajax 的众多优势，传统的 Web 开发受到了严重的挑战，越来越多的人也开始学习和采用 Ajax。

那么 Ajax 究竟在哪些方面要胜过传统 Web 呢？我们来看看下面的两张图，其中图 7-1 所示为传统 Web 应用程序的工作原理，图 7-2 所示为 Ajax 程序的工作原理。

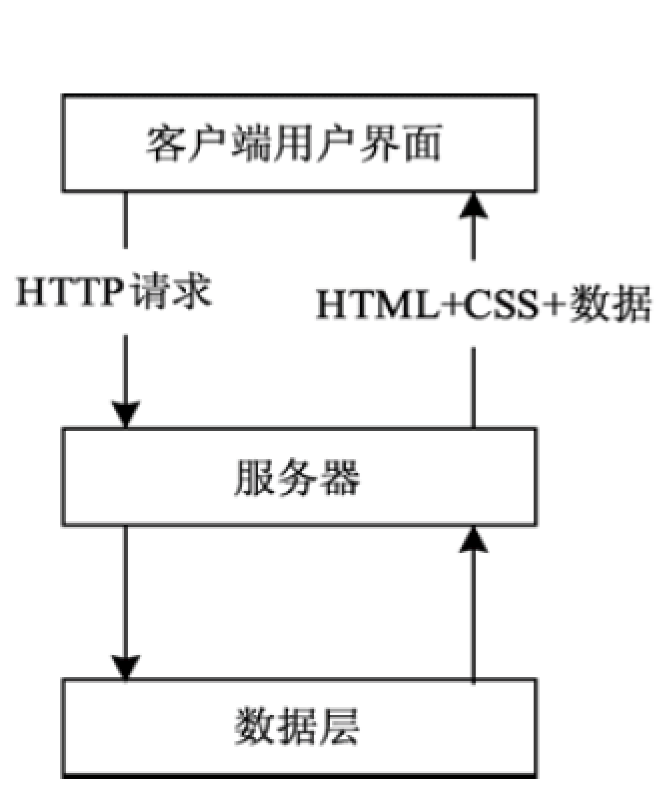


图 7-1　传统 Web 工作原理

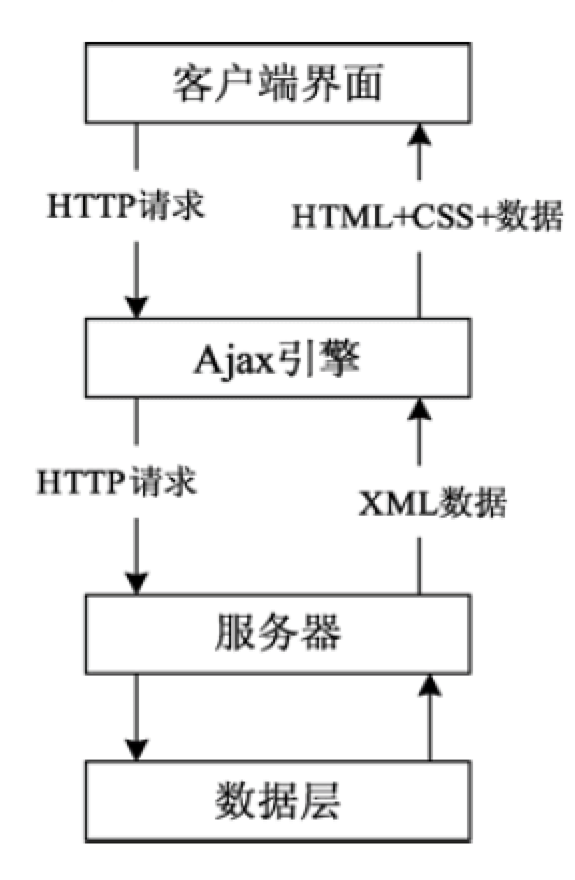


图 7-2　Ajax 工作原理

很明显，如图 7-1 所示的传统方式在提交请求时，服务器承担大量的工作，客户端只有数据显示的功能。而图 7-2 所给出的 Ajax 方式中，客户端界面和 Ajax 引擎都是在客户端运行，

这样大量的服务器工作可以在 Ajax 引擎处实现。

也就是说，与传统的 Web 应用不同，Ajax 采用异步交互过程。Ajax 在用户与服务器之间引入了一个中间介质，消除了网络交互过程中的处理和等待缺点。相当于在用户和服务器之间增加了一个中间层，使用户操作与服务器响应异步化。这把以前服务器负担的一些工作转移到客户端，利用客户端闲置的处理能力，来减轻服务器和带宽的负担，从而达到节约服务器空间以及降低带宽租用成本的目的。

2. XMLHttpRequest 对象

使用 Ajax 时，JavaScript 会调用 XMLHttpRequest 对象来直接与服务器通信。可见，XMLHttpRequest 对象是 Ajax 的核心。

Ajax 应用实现的框架和异步通信的架构，也是围绕 XMLHttpRequest 对象的创建、发送请求、处理响应来展开的。

XMLHttpRequest 对象并非最近才出现，最早在 Microsoft Internet Explorer 5.0 中被引入。目前，XMLHttpRequest 对象已得到大部分浏览器的支持，包括 Internet Explorer 5.0+、Safari 1.2、Mozilla 1.0 / Firefox、Opera 8+ 以及 Netscape 7。

例如，在 Internet Explorer 中可用如下简单的代码来创建一个 XMLHttpRequest 对象实例：

```
var httpRequest = new ActiveXObject("Microsoft.XMLHTTP");
```

XMLHttpRequest 对象提供了各种属性，以便于脚本处理和控制 HTTP 请求与响应。表 7-1 列出了处理 XMLHttpRequest 时常用的属性。

表 7-1　XMLHttpRequest 对象的属性

属　性	说　明
onreadystatechange	每个状态改变时都会触发这个事件处理器，通常会调用一个 JavaScript 函数
readyState	请求的状态
responseText	服务器的响应，表示为一个串
responseXML	服务器的响应，表示为 XML，这个对象可以解析为一个 DOM 对象
status	服务器的 HTTP 状态码(例如，200 对应 OK，404 对应 Not Found(未找到)，等等)
statusText	HTTP 状态码的相应文本(OK 或 Not Found 等)

XMLHttpRequest 对象提供了 send()、open()在内的 6 种方法，用来向服务器发送 HTTP 请求，并设置相应的头信息，表 7-2 列出了 XMLHttpRequest 对象的方法。

表 7-2　XMLHttpRequest 对象的方法

方　法	说　明
abort()	停止当前请求
getAllResponseHeaders()	把 HTTP 请求的所有响应首部作为键/值对返回
getResponseHeader(“header”)	返回指定首部的串值
open(“method”, “url”)	建立对服务器的调用，method 参数可以是 get、post 或 put，url 参数可以是相对 URL 或绝对 URL

续表

方　法	说　明
send(content)	向服务器发送请求
sendRequestHeader("header", "value")	把指定首部设置为所提供的值，在设置任何首部之前必须调用open()方法

随着本章的学习，读者将学习到 XMLHttpRequest 对象的更多知识，以及实例应用。

7.1.2　实例描述

众所周知，Ajax 的要点是 XMLHttpRequest 对象的使用。但是由于该对象不是一个 W3C 标准，因此虽然支持的浏览器很多，但不同浏览器下也是有差异的。

如此一来，想要在项目中使用 Ajax，首先要解决的头疼问题就是如何在多种浏览器下创建 XMLHttpRequest 对象。甚至可以说，如何很好地解决这一问题，在很大程度上决定了整个项目的开发效率和周期。

下面就来看看我们的最佳做法，它可以在项目需要创建 XMLHttpRequest 对象时，用一行代码就可以完成任务，而无须重复地做复制/粘贴工作。

7.1.3　实例应用

【例 7-1】创建跨浏览器的 XMLHttpRequest 对象。

我们先来分析一下，目前在支持 XMLHttpRequest 对象的浏览器下是如何创建该对象的。

- Internet Explorer 系列浏览器：把 XMLHttpRequest 实现为一个 ActiveX 对象。
- Firefox、Safari 和 Opera 浏览器：把 XMLHttpRequest 实现为一个本地 JavaScript 对象。
- 其他系列浏览器：把 XMLHttpRequest 实现为一个本地 JavaScript 对象。

根据这个原则得知，其实并不需要针对每个浏览器详细编写代码来区别浏览器类型。要创建一个 XMLHttpRequest 对象的实例，需要做的只是检查浏览器是否提供对 ActiveX 对象的支持。如果浏览器支持 ActiveX 对象，就可以使用 ActiveX 来创建 XMLHttpRequest 对象。否则，就要使用本地 JavaScript 对象技术来创建。

createXMLHttpRequest()函数实现了用 JavaScript 代码来编写跨浏览器的 XMLHttpRequest 对象实例：

```
var XmlHttp;
function createXMLHttpRequest()
{
    //在 IE 下创建 XMLHTTPRequest 对象
    try
    {
        XmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
    }
    catch(e)
    {
```

```
        try
        {
            XmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
        }
        catch(oc)
        {
            XmlHttp = null;
        }
    }
    //在 Mozilla 和 Safari 等非 IE 浏览器下创建 XMLHTTPRequest 对象
    if(!XmlHttp && typeof XMLHttpRequest != "undefined")
    {
        XmlHttp = new XMLHttpRequest();
    }
    if(!XmlHttp) alert("XMLHttpRequest 对象创建失败。");
    return XmlHttp;
}
```

7.1.4　实例分析



源码解析：

首先要创建一个全局作用域变量 XmlHttp 来保存这个对象的引用。创建 XMLHttpRequest 实例的具体任务由 createXMLHttpRequest 函数完成。在这个函数中只有简单的分支逻辑来确定如何创建对象。默认为调用 Internet Explorer 浏览器，通过生成 ActiveXObject 的一个新实例来创建 XMLHttpRequest 对象，并传入一个字符串指示要创建何种类型的 ActiveX 对象。在本示例中，为构造函数提供的字符串是 Microsoft.XMLHTTP，表示创建 XMLHttpRequest 的一个实例，然后返回这个对象，也可能返回 null。

如果调用失败，会尝试使用 Microsoft.XMLHTTP 来创建。最后，若两种情况均不成功，则说明使用的不是 Internet Explorer 浏览器，此时会把 XMLHttpReqest 实现为一个本地 JavaScript 对象并返回。

7.2　客户端验证注册表单完整性

我们平时在输入一些信息的时候会很随意，比如说用户名、密码或者邮箱什么的。而这些信息往往都有特定的格式限制，在提交到服务器后会根据输入的信息进行判断并给出反馈。

为了限制用户随意地提交表单带给服务器沉重的负担以及控制表单的完整性，我们引入了客户端表单验证的概念。



视频教学：光盘/videos/7/RegExp.avi　　　　长度：6 分钟

7.2.1 基础知识——正则表达式 RegExp 对象

表单验证的形式多种多样，编写客户端脚本对表单验证就是其中一种。这种方式的特点是完全在客户端、执行效率高、速度快、负载少，常见于各种注册表单。具体到实现上，体现为使用 JavaScript 结合正则表达式来完成。

JavaScript 提供了一个 RegExp 对象来完成有关正则表达式的操作和功能，每一条正则表达式模式对应一个 RegExp 实例。有两种方式可以创建 RegExp 对象的实例。

方法 1：

```
var str = "这是一个正则表达式"
var regex = new RegExp("一个");
```

方法 2：

```
var reg = /\d{2}-\d{5}/g;
var str = "ID is 654-6546549754";
var rs = reg.exec(str);
```

RegExp 对象有 3 个方法可以对正则表达式进行查找、匹配操作。

- exec()：该方法使用正则表达式模式在字符串中运行查找，并返回包含该查找结果的一个数组。
- test()：该方法返回一个布尔值，指出在被查找的字符串中是否存在与匹配模式相匹配的内容。
- match()：该方法使用正则表达式模式对字符串执行查找，并将查找的结果作为数组返回。

例如，下面编写的 CheckCreditNo()函数实现了通过正则表达式验证身份证号码的有效性：

```
function CheckCreditNo(num) {
    if (isNaN(num)) { alert("输入的不是数字！"); return false; }
    var len=num.length, re;
    if (len == 15)
        re = new RegExp(/^(\d{6})()?(\d{2})(\d{2})(\d{2})(\d{3})$/);
    else if (len == 18)
        re = new RegExp(/^(\d{6})()?(\d{4})(\d{2})(\d{2})(\d{3})(\d)$/);
    else { alert("输入的数字位数不对！"); return false; }
    var a = num.match(re);
    if (a != null) {
        if (len == 15) {
            var D = new Date("19" + a[3] + "/" + a[4] + "/" + a[5]);
            var B = D.getYear() == a[3]
              && (D.getMonth() + 1) == a[4]
              && D.getDate() == a[5];
        }
        else {
            var D = new Date(a[3] + "/" + a[4] + "/" + a[5]);
            var B = D.getFullYear() == a[3]
```

```
            && (D.getMonth() + 1) == a[4]
            && D.getDate() == a[5];
        }
        if (!B) { alert("输入的身份证号 "
          + a[0] + " 里出生日期不对！"); return false; }
    }
    return true;
}
```

上面的代码演示了 RegExp 对象的具体使用方法。在验证其他数据时，除了正则表达式有所变化之外，步骤都类似，这里就不再重复。

下面总结了一些在注册页面比较常用的正则表达式，并给出了详细的解释。

(1) 验证“用户名”正则表达式：

```
/^(\w){3,10}$/
```

\w：可以是字母，数字。{3,10}：用户输入的名字必须是大于 3 个且小于 10 个的字符。

(2) 验证“密码”的正则表达式：

```
/^[A-Za-z]{1}([A-Za-z0-9]|[._]){5,19}$/
```

^[A-Za-z]：第一个字符必须是大写或小写的字母。([A-Za-z0-9]|[._]){5,19}：可以是 5 个或 19 个字母、数字或下划线。

(3) 验证“日期”(即 1xxx-xx-xx)的正则表达式：

```
/^([1-9]/d{3})-((0?[1-9])|(1[0-2]))-((0?[1-9])|((1|2)[0-9])|30|31)$/
```

^([1-9]/d{3})：3 个 1 到 9 的数字。“-”：连接符号。((0?[1-9])|(1[0-2]))：用来限制月份，表示 0 可以出现零次或一次，后面跟着 1 到 9，或者出现 1，后面跟着 0 到 3 个数字。((0?[1-9])|((1|2)[0-9])|30|31)：限制天数，0 可以出现零次或一次，后面跟着 1 到 9，或者出现 1 或 2，后面跟着 0 到 9，或者是 30 天，31 天。

(4) 验证“电话号码”的正则表达式：

```
/^( (\d{3,4}-)|(\d{3,4}))\d{5,9}$/
```

^(\d{3,4}-)|(\d{3,4})在最前面是 3 到 4 个数字或者是 3 到 4 个数字再加一个“-”，后面是 5 个到 9 个数字。

(5) 验证“手机号码”的正则表达式：

```
^(\(\d{3}\))|(\d{3}\-))?13[0-9]\d{8}|15[89]\d{8}
```

^(\(\d{3}\))|(\d{3}\-))?：最前面可以是 3 个数字或 3 个数字和“-”。13[0-9]\d{8}|15[89]\d{8}：是以“13”开头，紧跟 0 到 9 的任意一个数字，后面是 8 个数字，或者以 158/159 开头，后跟 8 个数字。

(6) 验证“邮箱”的正则表达式：

```
/^[0-9a-zA-Z]+@[0-9a-zA-Z]+[\.]{1}[0-9a-zA-Z]+[\.]?[0-9a-zA-Z]+$/
```

^[0-9a-zA-Z]：在最前面可以是 0-10、a-z、A-Z 中的任意一个。“+”：表示前面的字符可以出现 1 次或多次。[\.]{1}：表示特殊字符“.”必须出现一次，而后面的“[\.]?”表示“.”可

以出现零次或一次。

(7) 验证“身份证号”的正则表达式：

```
/(^\d{15}$|(^\d{17})([0-9]|X)$)/
```

身份证号是 15 位数字或者是 17 位数字，最后一位是 0 到 9 之间的数字或“X”。

(8) 验证“邮编”的正则表达式：

```
/^[1-9]\\d{5}(?!\\d)/
```

^[1-9]第一位必须是 1 到 9 之间的任意数字，后面跟 5 位数。

(9) 验证“qq”号码的正则表达式：

```
/^[1-9][0-9]{4,13}/
```

^[1-9]：第一位的范围必须是从 1 到 9。[0-9]{4,13}：是 4 到 13 个任意数字。

7.2.2 实例描述

今天是阳光明媚、蓝天白云的好天气，心情也跟天气似的，非常不错。因为刚给客户顺利部署了一个项目，而且也得到了客户的认可。同时，再次证明了我们的实力——能够为客户提供人性化的用户体验和高质量的系统。

闲暇时，把项目的总结文档做出来了。发现正则表达式真是一个不错的东西，非常神奇。下面是项目中正则应用的冰山一角，一个注册表单的实现。

7.2.3 实例应用

【例 7-2】客户端验证注册表单完整性。

打开以前项目中有关注册的页面，把它另存为一个 HTML 文件，因为这里我们要在客户端实现验证。

调整一下 form 表单，对注册的信息进行精简，仅保留最“实用”的登录名、登录密码、确认密码以及电子邮箱 4 个项。如下所示为此时的 form 表单代码：

```
<form action="" method="post" id="apForm">
    <div class="content">
        <p>
            <label for="email"> 您的登录名：</label>
            <INPUT class="input" type="text" id=username maxLength=32
              onBlur= "checkusername();" name="username">
            <span id="checkusername"> </span> </p>
        <p>
            <label for="email"> 登录密码：</label>
            <INPUT  class="input"  style="width:148px;"
              onBlur= "checkpassword()" type=password maxLength=16
              id="password" name="userpass">
            <span id="checkpassword"> </span> </p>
```

```
        <p>
            <label for="email"> 密码确认：</label>
            <INPUT class="input" style="width:148px;"
              onBlur="checkpassword2()" type=password maxLength=16
              id="password2" name="userpass2">
            <span id="checkpassword2"> </span> </p>
        <p>
            <label for="email"> 电子邮箱：</label>
            <INPUT id=email maxLength=32 onBlur="checkemail()"
              class="input" name=email>
            <span id="checkemail"> </span> </p>
        <p>
            <INPUT type="hidden" name="registersubmit" value="true">
            <INPUT type="submit" name="mysubmit" value="" class="tj">
        </p>
    </div>
</form>
```

在这个表单之前添加 JavaScript 脚本，首先定义验证出错时的提示语。每种情况为一个变量，这种方式非常类似于常量，在需要时通过命名来进行引用，而无须每次都重复字符串。具体脚本如下：

```
<script type="text/javascript" language="javascript">
    var profile none = '请填写此项不能为空。'
    var profile username toolong =
      '对不起，您的用户名超过 15 个字符，请输入一个较短的用户名。';
    var profile username tooshort =
      '对不起，您输入的用户名小于 3 个字符，请输入一个较长的用户名。';
    var profile passwd notmatch = '两次输入的密码不一致，请检查后重试。';
    var profile email illegal = 'Email 地址无效，请重新填写。';
    var profile email invalid = '您只能使用以 结尾的信箱，请重新填写。';
    var profile email censor = '请不要使用以 结尾的信箱，请重新填写。';
    var doublee = parseInt('0');
    var lastusername = lastpassword = lastemail = '';
</script>
```

接下来，创建对登录名进行验证的代码，验证条件为：

- 内容不能为空，是必填项。
- 长度在 3~15 之间。

它在光标离开输入框时触发 checkusername()函数，该函数的实现代码如下：

```
function checkusername() {
    var cu = $('checkusername');
    var unlen = 0;
    var username = trim($('username').value);
    if(username == lastusername) {
        warning(cu, profile none);
        return;
    } else {
```

```
        cu.innerHTML = '<img src="images/check_right.gif" width="13" height="13">';
    }
    unlen = username.replace(/[^\x00-\xff]/g, "**").length;
    if(unlen<3 || unlen>15) {
        warning(cu, unlen < 3 ? profile username tooshort : profile username toolong);
        return;
    }
}
```

对密码的验证条件为检测不能为空，在光标离开时触发，实现函数 checkpassword()的代码如下：

```
function checkpassword(confirm) {
    var cp = $('checkpassword');
    var password = $('password').value;

    if(!confirm && password == lastpassword) {
        warning(cp, profile none);
        return;
    } else {
        lastpassword = password;
    }

    if(password == '' || /[\'\"\\]/.test(password)) {
        warning(cp, profile passwd illegal);
        return false;
    } else {
        cp.innerHTML = '<img src="images/check right.gif" width="13" height="13">';
        if(!confirm) {
            checkpassword2(true);
        }
        return true;
    }
}
```

登录密码为必填项，因此重复的确认密码也不能为空，且两者必须一致。
实现函数为 checkpassword2()，代码如下：

```
function checkpassword2(confirm) {
    var password = $('password').value;
    var password2 = $('password2').value;
    var cp2 = $('checkpassword2');

    if(password2 != '') {
        checkpassword(true);
    }
    if(password == '' || (confirm && password2 == '')) {
        cp2.style.display = 'none';
        return;
    }
```

```
    if(password != password2) {
        warning(cp2, profile passwd notmatch);
    } else {
        cp2.innerHTML = '<img src="images/check_right.gif" width="13" height="13">';
    }
}
```

再往下我们来看看如何对邮箱进行验证。邮箱地址本身就有格式限制，根据这个标准编写 checkemail()函数，最终产生的验证代码如下：

```
function checkemail() {
    var email = trim($('email').value);

    if(email == lastemail) {
        return;
    } else {
        lastemail = email;
    }

    var ce = $('checkemail');
    var accessemail = '';
    var censoremail = '';
    var accessexp = accessemail != '' ? /()$/i : null;
    var censorexp = censoremail != '' ? /()$/i : null;
    illegalemail = !(/^[\-\.\w]+@[\.\-\w]+(\.\w+)+$/.test(email));
    invalidemail = accessemail != '' ? !accessexp.test(email)
     : censoremail != '' && censorexp.test(email);
    if(illegalemail || invalidemail) {
        warning(ce, illegalemail ? profile email illegal :
         (accessemail != '' ? profile email invalid : profile email censor));
        return;
    }
    if(!doublee) {
        ce.innerHTML = '<img src="images/check right.gif" width="13" height="13">';
    }
}
```

7.2.4　运行结果

直接在浏览器中打开 HTML 文件，测试验证程序的正确性。可以在表单中看到上面设置的 4 项，先什么也不输，直接使光标离开输入框，可以看到验证不能为空的提示，如图 7-3 所示。

第 2 次测试从登录名开始，依次输入各种类型的内容，如图 7-4 所示为验证其有效性的结果。

经过一番测试后，发现我们的程序能够正确运行，并能够处理各种情况下的验证。这一步，我们把数据按正确格式填入，出现验证成功的结果，如图 7-5 所示。



图 7-3 验证不能为空



图 7-4 验证有效性



图 7-5 验证成功

7.2.5 实例分析



源码解析：

如果你先前一直使用的 ASP.NET 验证来完成这些工作。那本实例肯定给你带来了惊喜，因为这种方法实现太灵活了，而且显得非常轻便，脱离了 ASP.NET 服务器端控件的束缚。

另外，或许你已经注意到，实际上 ASP.NET 验证控件最终运行后也是生成对应的 JavaScript

代码。

实际上，正则表达式的作用远远不止这些，可以说是用于模式匹配和替换的强有力的工具。而本实例中的重点是如何在 JavaScript 中使用 RegExp 对象匹配正则表达式、验证结果以及对结果进行处理。

7.3　读取新闻列表 XML

如何减少对服务器的请求次数来节省性能，是每个网站开发都应该考虑的问题。通常的解决办法也就是将数据分别存储在不同位置，或者是增加硬件设备。

一种更好的解决方案是将 XML 作为存储模型发送到客户端，由于 XML 本身的优势及浏览器支持的特性，可以在客户端根据 XML 格式，直接进行读取并省去对数据库的请求。

 **视频教学：** 光盘/videos/7/SendXMLRequest.avi　　　　长度：8 分钟

7.3.1　基础知识——XML 操作

XML 是一种标准化的可以在 Web 上表示结构化信息的文本格式，利用它可以存储有复杂结构的数据信息。XML 是可以被任何计算机语言读取的标记语言，例如可以从 JavaScript 和 ASP.NET 等读取，同时也出现了很多与 XML 相关的技术，如 XSLT、DOM、XPath 等。

1. 获得 XML 对象

随着 XML 的流行，很多开发人员开始使用 JavaScript 编写一些自己的用于处理 XML 的对象。一些浏览器开发人员在感受到这些召唤后，开始大胆地向在客户端支持 XML 和 XML 相关语言的方向推进。其中，IE 浏览器在这方面做得比较完善。

例如，在 IE 中要创建这个对象的实例，使用以下代码：

```
var oXmlDom = new ActiveXObject("Microsoft.XmlDom");
```

执行这行代码后，oXmlDom 对象就同其他 DOM Document 的对象行为完全一样，表 7-3 中列出了 XML 对象的常用方法。

表 7-3　常用方法

名　称	说　明
createAttribute()	用指定的名称创建新的 attr 节点，即属性节点
createComment()	用指定的字符串创建新的 comment 节点
createElement()	用指定的标记名创建新的 element 节点
createTextNode()	用指定的文本创建新的 textNode 节点
getElementById()	返回文档中具有指定 id 属性的 element 节点
getElementsByTagName()	返回文档中具有指定标记名的所有 element 节点

续表

名　称	说　明
getChildNodes()	返回节点的所有子节点
appendChild()	追加一个子节点
getDocumentElement()	返回文档根节点
createCDATASection()	创建一个 CDATA 部分节点
createProcessingInstruction()	创建一个处理节点

2. XML 基本操作

XML DOM(XML Document Object Model、XML 文档对象模型)是用于获取、更改、添加或删除 XML 元素的标准。XML DOM 提供了用于操作 XML 的一系列接口，在使用时首先要理解：文档对象、节点对象和节点列表这几个概念。

为了帮助理解这些概念，这里举例说明。例如存在一个如下的 XML 文档：

```
<?xml version="1.0" encoding="utf-8" ?>
<books>
    <book>
        <name>ASP.NET 入门到精通</name>
        <author>汇智科技</author>
       <Publisher>清华大学出版社</Publisher>
    </book>
</books>
```

(1) 文档对象表示整个 XML 文档，也就是文档树的根，由它能获取 XML 的声明、文档类型等，对应 XML 对象。

(2) book 就是 XML DOM 中的一个节点对象，表示 XML 内部的单个对点，它是 Node 接口实现的对象，利用该对象我们可以完成对节点追加等方面的操作。

(3) 获取 book 的节点列表时就包含了 name、author 和 Publisher 这 3 项，它是子节点的集合，是由 NodeList 接口实现的对象，通过该对象我们可以对节点进行大批量的读取和操作。

一旦创建好 XML DOM 对象，就可以调用 loadXML()或 load()方法将 XML 文档载入。loadXML()方法可直接向 XML DOM 输入 XML 字符串：

```
var xmlDom = new ActiveXObject("Microsoft.XmlDom");
xmlDom.loadXML("<book><name>ASP.NET 入门到精通</name></book>");
```

在使用 load()方法载入 XML 文档时，可以分为两种模式：同步和异步。以同步模式载入文件时，JavaScript 代码会等待文件完全载入后才继续执行代码；以异步模式载入时，不会等待，可以使用事件处理函数来判断文件是否完全载入了。

默认情况下，load()方法按照异步模式载入。要进行同步载入，需要设置 async 属性值为 false。例如，通过 load()方法载入 book.xml 文档：

```
xmlDom.load(book.xml);
xmlDom.async = false;
```

把 XML 文档载入到 DOM 中后，肯定还需将 XML 文档内容读取出来。IE 为每个节点(包

括文档节点)都添加了一个 xml 特性，使得这个操作十分方便，它会将 XML 表现形式作为字符串返回，所以要获取载入后的 XML 十分简单。如下代码所示：

```
document.write(xmlDom.xml);
```

不仅可以获取整个 XML 文档的内容，还可以获取某个节点的内容。请看下面的代码：

```
var node = xmlDom.documentElement.childNodes[0].childNodes[0]; //name 元素
document.write(node.xml);
```

上面的代码看着是不是很熟悉啊。下面我们来跟着实例继续学习 XML 的旅程吧！

7.3.2　实例描述

凡事都具有双面性，即有好的一面，也有坏的一面。就拿数据持久化来说吧，在数据库中存储数据具有很多优势，但对于格式比较简单，量也比较少的数据，如果也用数据库就有点得不偿失，有大材小用的感觉。

下面的实例介绍了最佳替代方案——XML 的使用，它具有速度快、支持广泛等特点。实例直接在客户端利用 JavaScript 读取服务器上的 XML 来完成。

7.3.3　实例应用

【例 7-3】读取新闻列表 XML。

(1)　在网站的根目录下创建一个 XML 文档，向其中添加数据。这里为新闻列表数据，包含 4 个新闻分类，并定义每个分类下的新闻，如新闻标题、发表人和时间。最终的格式如下：

```
<?xml version="1.0" encoding="gb2312" ?>
<itZcn>
   <type name="1">窗内日志
      <item>
         <title>美女进来报道，帅哥别进来了</title>
         <user>somboy</user>
         <fdate>09-12-10</fdate>
      </item>
      ...
   </type>
   <type name="2">热门分享
      ...
   </type>
   <type name="3">焦点视频
      ...
   </type>
   <type name="4">技术文档
      ...
   </type>
</itZcn>
```

可以看到，itZcn 为 XML 的根节点；type 是其子节点；item 是 type 下子的节点，是 itZcn 的孙节点。每个 item 代表一个新闻，里面又包含了 title、user 和 fdate 这 3 个子节点。

(2) 此时的 XML 文件将被作为数据库来使用，用于存放新闻。这里打算使用客户端 JavaScript 来读取 XML，先在页面中设计布局，再添加下面的代码显示新闻列表：

```
<div class="tabList" id="tabList1">
    <ul class="tabBox">
    <script>
    showTabBars();                                         //显示所有类别
    document.getElementById("crc06_1").className = "tabOption tabOn";//初始化
    </script>
    </ul>
</div>
```

(3) 在页面中添加上面调用的 showTabBars()函数，它的主要作用是从 XML 中获取所有的新闻类别并显示：

```
function showTabBars()
{
    var doc;
    //创建 XML DOM 对象
    if (window.ActiveXObject) { //IE 浏览器下创建
        var doc = new ActiveXObject("Microsoft.XMLDOM");
        doc.async = "false";       //设置为同步
        doc.load("XMLFile.xml"); //载入 XMLFile.xml 文件
    }
    else {                         //非 IE 浏览器下创建
        var parser = new DOMParser();
        var doc = parser.parseFromString("XMLFile.xml", "text/xml");
    }
    var tabs =
      doc.getElementsByTagName('type'); //获取 XML 中的所有 type 元素到 tabs 数组
    for (i=0; i<tabs.length; i++) {                     //循环遍历 tabs 数组
        var s1 = tabs[i].getAttribute('name');       //name 属性
        var s2 = tabs[i].childNodes[0].nodeValue;    //类别
        document.write(makeTabBar(s1, s2));      //输出分类
    }
}
```

利用 XML DOM 的 getElementsByTagName()方法获取了所有 type 元素，并保存到 tabs 数组中。数组中的每个元素均为一个 Node 对象，通过 getAttribute('name')获取 type 的 name 属性，childNodes[0].nodeValue 则获取了第 1 个子节点的值，即新闻类型。最后的 makeTabBar()函数则将输出一段 HTML 代码。

(4) 编写的 makeTabBar()函数具有两个参数，第 1 个是一个用于标识的 ID，第 2 个则是要显示的类别名称，最终返回组织后的 HTML 代码：

```
function makeTabBar(tId, tName)
{
```

```
    var html =
      "<li class=\"tabOption\" id=\"crc06 " + tId + "\">" //用 id 标识类别开始
      + "<h4><a href=\"#\"><span>"
      + tName + "</span></a></h4>"  //显示新闻类别名称
      + "<div class=\"tabContentBox \"> "
      + makeTabContent(tId)          //显示该类下的新闻
      + "</div>"
      + "</li>";                     //类别结束

    return html;
}
```

(5)　其实在显示类别时，就同时获取了该类下的新闻信息，这就是 makeTabContent()函数的作用。它的参数 tId 为要获取新闻的类别编号，此处在调用时指定：

```
function makeTabContent(tId)
{
    var ht = "<dl class=\"tabContent blog\">";  //新闻列表开始
    //创建 XML DOM 对象
    if (window.ActiveXObject) {
        var doc = new ActiveXObject("Microsoft.XMLDOM");
        doc.async = "false";
        doc.load("XMLFile.xml");
    }
    else {
        var parser = new DOMParser();
        var doc = parser.parseFromString("XMLFile.xml", "text/xml");
    }
    var x = doc.getElementsByTagName('type');
    for (var i=0; i<x.length; i++) {
        var s1 = x[i].getAttribute('name');  //得到 type 节点的 name 属性值
        if (s1 == tId) {                     //判断是否为指定的 ID
            var childnode = x[i].getElementsByTagName('item');
            for (var j=0; j<childnode.length; j++) {
                node1 =
                  childnode[j].childNodes[0].firstChild.nodeValue; //新闻名称
                node2 =
                  childnode[j].childNodes[1].firstChild.nodeValue; //发表人
                node3 = childnode[j].childNodes[2].firstChild.nodeValue; //时间
                ht += "<dt><a href=\"#\" >" + node1 + "</a></dt>"
                  + "<dd class=\"name\"><a href=\"#\">" + node2 + "</a></dd>"
                  + "<dd>" + node3 + "</dd>";
            }
            ht += "</dl>";              //新闻列表结束
        }
    }
    return ht;                          //返回整个新闻列表
}
```

这里先判断是否为要显示的 ID，如果相等则继续。getElementsByTagName('item')方法返回的是 XML DOM 的 NodeList 对象，对应 XML 中的 item 元素集合，表示具体的新闻信息。最后，经过一番的 HTML 代码组织，形成有效的布局，再用 ht 变量返回给调用者。

7.3.4 运行结果

将 HTML 与 XML 放在同一级目录，然后在 IE 浏览器中打开 HTML 文件。可以看到默认运行后会打开第 1 个标签，显示新闻列表的效果，如图 7-6 所示。



图 7-6 查看默认新闻效果

每一个类别为一个标签，当鼠标移到上面时将自动切换至该类下的新闻，如图 7-7 所示。



图 7-7 切换类别标签

7.3.5　实例分析



源码解析：

在本实例中没有对 XML 本身知识的过多讲解。这里的 XML 文档存在 4 层嵌套关系，依次为 itZcn→type→item→title/user/fdate。

所以，先利用 getElementsByTagName()方法指定节点名称来获取元素集合。通过 length 属性得到集合的长度并循环，循环时再对节点的属性和值进行操作，并与预先定义的 HTML 布局进行组合，形成最终代码并返回。

实例中通过函数封装功能，使代码看得非常简洁。其中，showTabBars()是入口函数，调用此函数来显示效果；makeTabBar()函数则用来显示新闻类别，即生成标签；第 3 个函数 makeTabContent()对所有的新闻进行显示，并生成列表。

7.4　发送异步请求获取服务器时间

同步请求的使用要比异步请求简单，因为它们将直接返回数据，并免除了创建回调函数的麻烦。但是，这并不是使用 XMLHttpRequest 的标准方法，因为在请求发生时浏览器会被锁定。在有些情况下，这种阻塞是很有效的(主要是当决策必须在当前函数结束之前做出时)。

但在大多数情况下，会希望这些请求在后台完成。而异步请求则允许浏览器在载入新数据时继续执行 JavaScript 代码，用户也可以继续与页面进行交互。加上适当的用户界面，异步通信机制能够使得 Ajax 应用程序十分有用，即使用户与该网站的连接速度很慢。



视频教学：光盘/videos/7/XHRObjectLiveCycle.avi　　长度：6 分钟

7.4.1　基础知识——XMLHttpRequest 对象的运行周期

Ajax 程序主要通过 JavaScript 事件来触发，在运行时需要调用 XMLHttpRequest 对象发送请求和处理响应，客户端处理完响应之后，XMLHttpRequest 对象就会一直处于等待状态，这样一直周而复始地工作。

Ajax 实质上是遵循“客户端/服务器端”模式，所以这个框架的基本流程是：XMLHttpRequest 对象初始化→发送请求→服务器接收→服务器返回→客户端接收→修改客户端页面内容。只不过这个过程是异步的，其周期如图 7-8 所示。

在图 7-8 中，Ajax 中间层显示了 XMLHttpRequest 对象的运行周期。

(1)　当 Ajax 中间层从客户端界面获取请求信息之后，需要初始化 XMLHttpRequest 对象。

(2)　初始化完成之后，通过 XMLHttpRequest 对象将请求发送给服务器端。

(3)　服务器端获取请求信息后，处理并返回响应信息。

(4)　然后 Ajax 中间层获取响应，通过 XMLHttpRequest 对象将响应信息和 Ajax 中间层所

设置的样式信息进行组合，即处理响应。

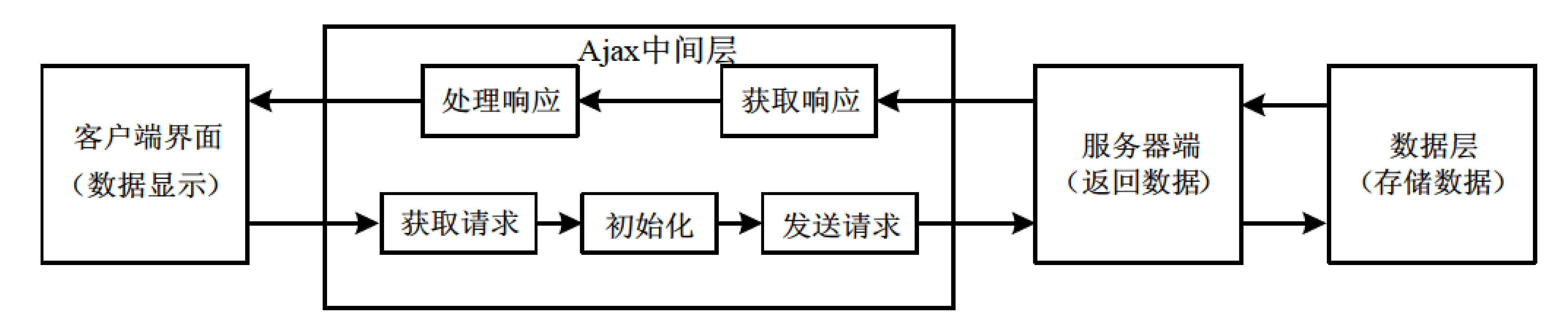


图 7-8　XMLHttpRequest 对象的运行周期

(5)　最后 Ajax 中间层将所有的信息发送给客户端界面，并显示由服务器返回的信息。

7.4.2　基础知识——open()方法

open()方法的完整语法格式如下：

```
open(string method, string uri, boolean async, string username, string password)
```

其中，method 是必选参数，用于指定用来发送请求的 HTTP 方法(GET、POST、PUT、DELETE 或 HEAD)。为了把数据发送到服务器，应该使用 POST 方法；但若要从服务器端检索数据，应该使用 GET 方法。

另外，uri 参数用于指定 XMLHttpRequest 对象把请求发送到的服务器相应的 URI。async 参数指定是否请求是异步的，默认值为 true，在发送同步请求时需要把这个参数设置为 false。对于需要认证的服务器，可以提供可选的用户名和口令参数。

(1)　在调用 open()方法准备好一个请求之后，还需要把该请求发送到服务器。仅当 readyState 值为 1 时，才可以调用 send()方法；否则 XMLHttpRequest 对象将引发一个异常。该请求被使用提供给 open()方法的参数发送到服务器。

(2)　当 async 参数为 true 时，send()方法立即返回，从而允许其他客户端脚本处理继续。在调用 send()方法后，XMLHttpRequest 对象把 readyState 的值设置为 2。

(3)　当服务器响应时，在接收消息体之前，如果存在任何消息体的话，XMLHttpRequest 对象将把 readyState 设置为 3。

(4)　当请求完成加载时，它把 readyState 设置为 4。对于一个 HEAD 类型的请求，它将在把 readyState 值设置为 3 后再立即把它设置为 4。

7.4.3　实例描述

日期和时间的处理不仅在面试题中会考问，在实际项目开发中也是我们经常需要处理的问题，似乎没有哪个项目可以避开它们，我们常常在处理用户的出生年月日、注册日期、订单的创建时间等属性时用到，由此可见其重要性。

提到日期和时间，大家最先想到的是当前的时间与日期。没错，下面我们就使用最新 Ajax 技术来实战一下，看看它是如何实现获取服务器上当前时间的……

7.4.4 实例应用

【例 7-4】发送异步请求获取服务器时间。

在开始编写代码之前，我们首先找一个 HTML 页面，并准备一个位置来显示时间：

```
当前时间：<span id="sDate" onclick=""></span>
```

由于 Ajax 是运行在客户端的，这一步添加 JavaScript 代码来实时获取时间，达到不刷新的效果：

```
<script language="javascript" type="text/javascript">
setInterval("getTime()", 1000);
</script>
```

接上方的代码编写所需的 getTime()函数。使用 Ajax 时 XMLHttpRequest 对象是必须的，在函数体内调用 createXMLHttpRequest()函数来完成：

```
//发送 GET 请求
function getTime() {
    //创建 XMLHttpRequest 对象
    createXMLHttpRequest();
}
```



createXMLHttpRequest()函数的代码见本章的 7.1.3 小节，它可以创建一个跨浏览器的 XMLHttpRequest 对象。

有了对象之后，也就相当于有了主角。接下来，需要指定主角的工作地点，也就是指定一个服务器路径。此处设置为站点根目录下的 Server.aspx 文件：

```
//定义一个变量获取 ASPX 的路径
var url = "/Server.aspx?t=" + Math.random();
```

然后指定当服务器返回信息时客户端的处理方式。这就需要将相应的处理函数名称赋给 XMLHttpRequest 对象的 onreadystatechange 属性：

```
XmlHttp.onreadystatechange = handleStateChange;
```

handleStateChange 表示是一个函数，我们将稍后介绍。注意，这里指定时不需要加括号，也不带参数。

现在我们就可以向服务器发送异步请求了，方法是调用 open()和 send()方法：

```
XmlHttp.open("GET", url, true);
XmlHttp.send(null);
```

这一步之后 getTime()函数就编写完成了。此时，Ajax 的核心对象 XMLHttpRequest 将会与指定的服务器建立连接，并发送 GET 请求。在得到服务器的响应结果之后，将转交给 onreadystatechange 属性指定的响应函数进行处理。

创建 handleStateChange()函数，在函数中调用 XMLHttpRequest 对象的 responseText 属性获取返回结果。再通过 JavaScript 代码直接插入到前台设置的 HTML 位置中：

```
function handleStateChange() {
    if (XmlHttp.readyState == 4) {
        //判断对象状态
        if (XmlHttp.status == 200) {
            document.getElementById("sDate").innerHTML = XmlHttp.responseText;
        }
    }
}
```

最后，我们来看看在 ASPX 服务器端究竟返回了什么给客户端。其实就是一个包含当前日期和时间的字符串，在页面加载后直接输出：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.Write(DateTime.Now.ToString("yyyy-MM-dd HH:mm:ss"));
}
```

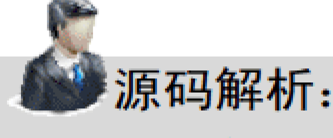
7.4.5 运行结果

从 Web 服务器上浏览 HTML 页面，它将对同目录下的 Server.aspx 页面进行请求。运行后会看到不断更新的时间实时跳动，如图 7-9 所示。



图 7-9 获取服务器时间的运行效果

7.4.6 实例分析



源码解析：

可以看到，本实例中，服务器端虽然只有一行代码，但在客户端编写的代码量却很多。这

也再次说明了 Ajax 侧重于客户端用户体验的特点，将大量需要处理的业务逻辑放在客户端，服务器仅处理重要数据与结果。

在整个实例的制作过程中，都是围绕 XMLHttpRequest 对象展开的，像创建该对象、指定请求 URL、发送请求、指定处理结果函数等。

由此可见 XMLHttpRequest 对象是实现 Ajax 应用必不可少的核心技术，负责 Ajax 与服务器的异步交互。

XMLHttpRequest 对象的使用方法并不复杂，随着本章内容的深入，我们将介绍更多有关该对象的知识。

7.5　验证用户名是否已经存在

输入校验是我们经常遇到的问题，这种问题很多时候是可以在 JS 里解决。但有些时候却需要访问后台，如在申请用户的时候检查用户名是否重复等问题。那就试试用 Ajax 吧，用它页面不会有刷新。

 **视频教学：** 光盘/videos/7/XHRObject.avi　　　　长度：4 分钟

7.5.1　基础知识——readyState 和 status 属性

1. readyState

作者昨天做了一个利用 Ajax 实现页面无刷新地从服务器端获得时间的例子程序，当时对于 xmlhttprequest 对象的 readystate 的后 3 个状态不是很清楚，想了半天，也不明白！

后来在程序的不同地方中用 alert(xmlhttp.readystate)查看 readystate 值的变化，并且请教朋友之后，才弄明白了其中的几个问题。

创建 xmlhttprequest 对象之后没有调用 open 之前 readystate 值为 0。调用 open()之后就变为 1 了，并且此时 onreadystatechange 函数与 open()几乎是同时执行的。接下来调用 send 方法后，在 startHttpRequest 函数中 readystate 值仍为 1，而调用 send 方法后应该有 2，3，4 三个状态，而只有在 startHttpRequest 函数用 alert 语句才可以观察到 3 个值！这是为什么呢？这是因为在 startHttpRequest 函数中当解析到 send 这一句时，并没有真正开始执行 send。

只有 send 执行，才可以在 onreadystatechange 函数观察到状态值的变化。readystate 不是发送的状态，它是准备发送的状态，要把它想象成人间大炮的“一级准备、二级准备、放”这样的口号，而不是请求发送本身。同时 xmlhttp 也不是监听服务器信息，它是在 send 的时候获取服务器返回的状态信息而已，只有一次，监听则是一直在观察状态。

至此，心中的疑惑全部解开！关于 readystate 的 5 个状态总结如下。

- (0)：未初始化。此阶段确认 XMLHttpRequest 对象是否创建，并为调用 open()方法进行初始化做好准备。值为 0 表示对象已经存在，否则浏览器会报错——对象不存在。
- (1)：载入。此阶段对 XMLHttpRequest 对象进行初始化，即调用 open()方法，根据参数(method, url, true)完成对象状态的设置。并调用 send()方法开始向服务端发送请求。

值为 1 表示正在向服务端发送请求。

- (2)：载入完成。此阶段接收服务器端的响应数据。但获得的还只是服务端响应的原始数据，并不能直接在客户端使用。值为 2 表示已经接收完全部响应数据，并为下一阶段对数据解析做好准备。
- (3)：交互。此阶段解析接收到的服务器端响应数据。即根据服务器端响应头部返回的 MIME 类型把数据转换成能通过 responseBody、responseText 或 responseXML 属性存取的格式，为在客户端调用做好准备。状态 3 表示正在解析数据。
- (4)：完成。此阶段确认全部数据都已经解析为客户端可用的格式，解析已经完成。值为 4 表示数据解析完毕，可以通过 XMLHttpRequest 对象的相应属性取得数据。

概括之，整个 XMLHttpRequest 对象的生命周期应该包含如下阶段：

```
创建→初始化请求→发送请求→接收数据→解析数据→完成
```

2. status

虽然常写 Ajax 方面的东西，但是很少去专门了解 xmlHttpRequest 的 status 各个值的含义，只是在用到时 Google 一下，下面将他人已记录过的总结一下，学习学习。

xmlHttpRequest 对象的 status 代表当前 http 请求的状态，是一个长整型数据，现在介绍一下它的含义。

1XX——此状态代码表示临时的响应。客户端在收到常规响应之前，应准备接收一个或多个 1xx 响应：

```
100 - 初始的请求已经接受，客户应当继续发送请求的其余部分。
101 - 服务器将遵从客户的请求转换到另外一种协议。
```

2XX——这类状态代码表明服务器成功地接受了客户端请求：

```
200 - OK。一切正常，对 GET 和 POST 请求的应答文档跟在后面。
201 - Created。服务器已经创建了文档，Location 头给出了它的 URL。
202 - Accepted。已经接受请求，但处理尚未完成。
```

3XX——客户端浏览器必须采取更多操作来实现请求。例如，浏览器可能不得不请求服务器上的不同页面，或通过代理服务器重复该请求：

```
300 - Multiple Choices。客户请求的文档可以在多个位置找到，这些位置已经在返回的文档内列出。如果服务器要提出优先选择，则应该在 Location 应答头指明。
301 - Moved Permanently。客户请求的文档在其他地方，新的 URL 在 Location 头中给出，浏览器应该自动地访问新的 URL。
302 - Found。类似于 301，但新的 URL 应该被视为临时性的替代，而不是永久性的。
303 - See Other。类似于 301/302，不同之处在于，如果原来的请求是 POST，Location 头指定的重定向目标文档应该通过 GET 提取。
```

4XX——发生错误，客户端似乎有问题。例如，客户端请求不存在的页面，客户端未提供有效的身份验证信息：

```
400 - Bad Request。请求出现语法错误。
404 - Not Found。无法找到指定位置的资源。这也是一个常用的应答。
405 - Method Not Allowed。请求方法对指定的资源不适用，用来访问本页面的 HTTP 谓词不被允许(方法不被允许)。
```

```
411 - Length Required。服务器不能处理请求，除非客户发送一个 Content-Length 头。
412 - Precondition Failed 请求头中指定的一些前提条件失败(HTTP 1.1 新)。
414 - Request URI Too Long URI 太长(HTTP 1.1 新)。
415 - 不支持的媒体类型。
417 - 执行失败。
423 - 锁定的错误。
```

5XX——服务器由于遇到错误而不能完成该请求：

```
500 - Internal Server Error。服务器遇到了意料不到的情况，不能完成客户的请求。
501 - Not Implemented。服务器不支持实现请求所需要的功能，页眉值指定了未实现的配置。
503 - Service Unavailable。服务不可用，服务器由于维护或者负载过重未能应答。
504 - Gateway Timeout。网关超时，由作为代理或网关的服务器使用，表示不能及时地从远程服务器获得应答。
505 - HTTP Version Not Supported。服务器不支持请求中所指明的 HTTP 版本。
```

7.5.2　实例描述

注册登录是许多网站用户管理模块不可缺少的一部分。注册时接收用户提交的注册信息，并进行一定的验证，最后把合法的数据存储到数据库中。其中最重要的就是对用户名(用户账号)进行验证，因为通常情况下用户名是区别每一个用户的唯一标志，它是不允许出现重复的。

在传统 Web 应用程序中，用户注册验证是用户操作体验相当差的一部分。因为，为了验证某一账号是否已经被人注册过，就必须等待注册页面被提交后返回的信息，或者通过单击某一“查看该用户是否可用”按钮，从弹出窗口中得到信息来确认账号的可用性。这一切是相当麻烦的，有些处理不好的传统 Web 应用程序，可能因为某一用户名已经使用，再返回用户注册页面时可能导致用户先前输入的用户注册信息也全部丢失，不得不再从头输入用户信息。这种情况的出现让用户很是反感。而现在通过采用 Ajax 技术，已经极大地改善了这个过程的用户操作体验。

我们将通过一个用户注册验证的实例来验证一下通过采用 Ajax 技术是如何改善用户操作体验的。该实例用于通过对用户名(用户账号)进行验证，也就是根据该用户名是否已经使用，在不进行页面刷新的情况下给出相应的提示信息。

7.5.3　实例应用

【例 7-5】验证用户名是否已经存在。

(1) 在 HTML 页面中添加一个表单，为表单设置注册时所要填写的信息。这里的注册页面很简单，包括用户名、密码、确认密码及邮箱 4 项。部分代码如下：

```
<FORM action = ""  name = "testForm" method = "post" onSubmit="return subTest()">
<table width="90%" align="center" cellpadding="5" cellspacing="1" >
  <tr valign="bottom">
    <td colspan="3" ><a  href="#">会员注册</a></td>
  </tr>
  <tr >
```

```
    <th width="23%">用户名</th>
    <td width="51%" align="left"><input type="text"  name = "userName"
      id="userName" onblur="checkNameInfo()" /></td>
    <td width="26%" align="left">
      <span id="nameinfo">*长度大于 4,小于 10</span>
    </td>
  </tr>
  <tr>
    <th>密码</th>
    <td><input name = "userPassword" type = "password" /></td>
    <td><span>*必填</span></td>
  </tr>
  <tr>
    <th>验证密码</th>
    <td><input type = "password" name = "reUserPassword" /></td>
    <td><span>*必填</span></td>
  </tr>
  <tr>
    <th>邮箱验证</th>
    <td><input type = "text"  name = "Email" id="mailaddress"  /></td>
    <td><span>*必填</span></td>
  </tr>
  <tr>
    <td> </td>
    <td><input name="提交" type = "submit" value = "登录", />
      <input name="重置" type = "reset" value="重置" /></td>
    <td> </td>
  </tr>
</table>
</form>
```

在上述表单中，用于输入用户名的文本框 id 属性为 userName，onblur 属性指定在文本框失去焦点事件触发时触发 checkNameInfo()事件处理程序。checkNameInfo()使用 Ajax 异步处理，并在 id 为 nameinfo 的标记中显示结果。

(2) 在页面的<head>标记中，创建 XMLHttpRequest 对象的 createXMLHttpRequest()函数。添加 checkNameInfo()函数将输入的用户名作为参数发送到 check.asp，并指定回调函数。代码如下所示：

```
function checkNameInfo() {
    var s = document.getElementById("userName").value;
    var url = "check.aspx?uname=" + escape(s);
    createXMLHttpRequest();
    XmlHttp.onreadystatechange = handleStateChange;
    XmlHttp.open("GET", url, true);
    XmlHttp.send(null);
}
```

上述代码中，语句 XmlHttp.open(“GET”, url, true)就是用于建立对服务器的调用，其中 url

此时的值是传递过来的“check.aspx?uname=”+escape(s)。可以看到指定 check.aspx 来处理请求，处理完成后使用 handleStateChange()来回调。

(3) 在用户注册页面所在的目录中新建一个 check.aspx 文件，并添加如下代码的实现代码：

```
protected void Page Load(object sender, EventArgs e)
{
    Response.Charset = "utf-8";
    string dburl = Server.MapPath("app data/Student.mdb");
    OleDbConnection conn = new OleDbConnection(
      "PROVIDER=Microsoft.Jet.OLEDB.4.0;DATA Source=" + dburl);
    string uName = Request["uname"];
    string strSQL = "select * from [user] where name='" + uName + "'";
    OleDbCommand cmd = conn.CreateCommand();
    cmd.CommandText = strSQL;
    conn.Open();
    OleDbDataReader dr = cmd.ExecuteReader();
    if (dr.Read())
    {
        Response.Write("1");
    }
    else
    {
        Response.Write("0");
    }
    conn.Close();
    Response.End();
}
```

在上述代码中，通过 uName = Request[“uname”]获取传递过来的用户名，然后进行验证，并根据验证结果输出相应的值。这里检验时使用了一个简单的 Access 数据库 Student.mdb，其中的 user 为会员表，通过传递的值在表里检索，如果找到匹配的，说明已存在相同的，输出 1；否则输出 0，最后关闭数据库连接。

(4) 在客户端使用 XmlHttp 发送请求后，每次状态改变都会调用 onreadystatechange 属性的 handlStateChange 函数。该函数的代码如下：

```
function handleStateChange() {
    var nameinfo = document.getElementById("nameinfo");
    nameinfo.innerHTML = "正在查询请稍候...";
    if (XmlHttp.readyState == 4) {
        if (XmlHttp.status == 200) {
            //显示结果
            var tmp = XmlHttp.responseText;
            if (tmp == 1) {
                nameinfo.innerHTML = "用户名已经存在，请选择其他名称。";
            }
            else {
                nameinfo.innerHTML = "用户名未被注册，可以使用。";
            }
```

```
            }
        }
}
```

7.5.4　运行结果

在浏览器中打开会员注册页面，在表单中输入一个用户名。一旦光标离开时就会建立请求，显示“正在查询请稍候...”的提示信息，如图 7-10 所示。当请求完成并返回后将显示相应的提示信息。例如，如图 7-11 所示是返回 0，为可以注册时的效果。



图 7-10　发送请求时

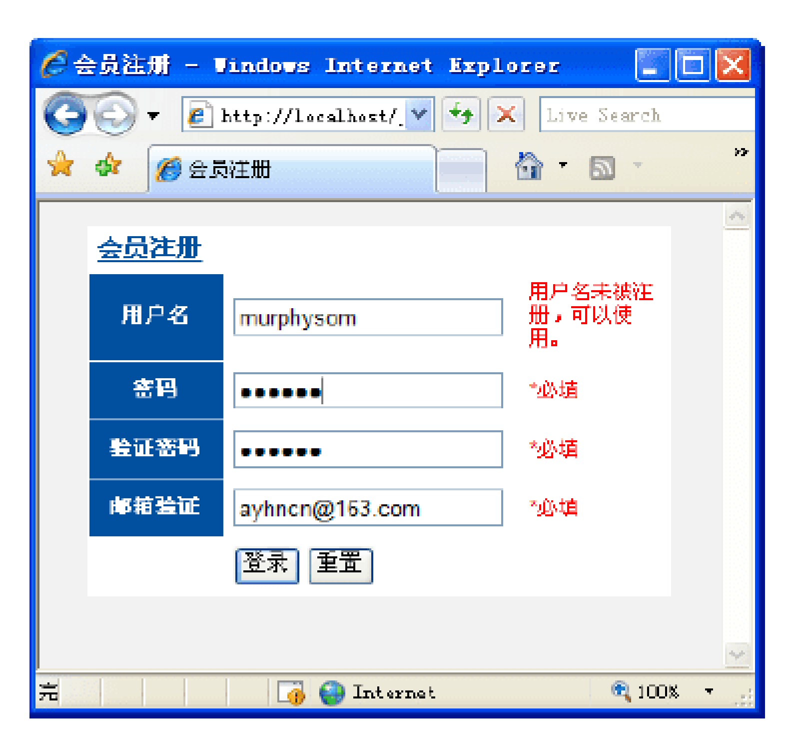


图 7-11　处理结果

7.5.5　实例分析



源码解析：

实例的重点仍然在客户端上，在服务器端仅仅是根据传递的用户名，对数据库进行检查，并返回一个简单标识。客户端根据这个标识，提示该用户名是否可用，还是处于查询状态。

onblur 事件使得用户在每次输入完成后都会进行验证处理。服务器端使用 Access 数据库进行用户名的匹配判断，客户端则根据 XMLHttpRequest 对象的 readystate 属性和 status 属性判断要显示的提示信息。

整体来说，实例的难点也在这两个属性身上，灵活掌握它们，可以轻松编写出各种验证功能。

7.6　添加国家代码表

HTTP 协议下的两大数据传输方式：GET 和 POST，它们各有所长。在大型的 Web 项目中，基于安全角度的考虑，数据会选择以 POST 方式进行传输。Ajax 的默认方式也是 POST，可见

它是如何受欢迎，它与 GET 方式唯一不同的就是数据的发送位置。

下面我们来详细看一下具体的实现代码。

 **视频教学：**光盘/videos/7/SendArgument.avi　　　　　长度：8 分钟

7.6.1　基础知识——send()方法

典型的应用是使用它并通过 POST 方法把数据发送到服务器。另外，也可以显式地调用无参数 send()方法，对于大多数情况，在调用 send()方法之前应该使用 setRequestHeader()方法先设置 Content-Type 头部。如果在 send(data)方法中的 data 参数的类型为 string，那么数据将被编码为 UTF-8。

例如，向 Server.aspx 文件以异步方式发送一个 GET 请求，便可以使用如下的代码：

```
xmlHttp.open("GET", "Server.aspx", true);
xmlHttp.send(null);
```

现在，同样需要向 Server.aspx 文件发送请求，不同的是要求请求中带有一些参数字符串。实现时有两种方法，第一种使用 open()方法在 URL 参数指定：

```
xmlHttp.open("GET", "Server.aspx?name=zht&pwd=123&mail=abc@163.com", true);
xmlHttp.send(null);
```

第二种是使用 send()方法指定：

```
xmlHttp.open("POST", "Server.aspx", true);
xmlHttp.send("name=zht&pwd=123&mail=abc@163.com");
```

7.6.2　实例描述

你能说出与国家有关的至少 3 个属性吗？

相信大多数人知道的只有国家中文名称和英文名称。原本对此了解也不多，直到经过最近的一个外包项目，才得到了完整的答案。

这是一个跨国公司的医疗保健项目，当时作者的小组负责的是基础信息的管理模块。其中有一个需求就是根据标准 GB/T2659—2000 记录国家代码，它将作为器械生产厂商、药品生产地、机构地址等信息的参考依据。

下面揭晓刚才问题的答案，其实在 GB/T2659—2000 文献中有详细的记录，包括国家 3 位数字代码、2 位英文字母代码、3 英文字母代码、中文名称、中文简称、英文名称和英文简称，共 7 个。

趁热打铁，这就开工，先从国家代码表的添加开始……

7.6.3　实例应用

【例 7-6】添加国家代码表。

创建一个 HTML 页面，然后添加国家代码表的录入表单。实例中的最终布局效果如图 7-12 所示。

图 7-12　国家代码表的录入表单

既然是使用 Ajax，XMLHttpRequest 对象的创建肯定是少不了的。这里就不再重复了，可参考上面例子中的实现。

图 7-12 中“提交”按钮单击后执行 sendData()函数，它是 Ajax 的主函数，实现代码如下：

```
function sendData() {
    createXMLHttpRequest();
    var url = "/sCountry.aspx?t=" + Math.random();
    XmlHttp.open("POST", url, true);
    XmlHttp.onreadystatechange = handleStateChange;
    XmlHttp.setRequestHeader(
      "Content-Type", "application/x-www-form-urlencoded;");
    XmlHttp.send(createDataString());
}
```

可以看到，open()方法指定使用 POST 作为传输方式。在 send()方法中指定了一个参数，createDataString()函数可返回要发送的参数数据。

创建 createDataString()函数，它的代码比较简单，就是按照一定的格式组装成一个大字符串后返回：

```
function createDataString() {
    var CountryID1 =
      document.getElementById("countryID").value;        //3 位数字代码
    var CountryID2 =
      document.getElementById("countryID2").value;       //2 位英文代码
    var CountryID3 =
      document.getElementById("countryID3").value;       //3 位英文代码
    var Name_ch = document.getElementById("name_ch").value;     //中文名称
    var Name_en = document.getElementById("name_en").value;     //英文名称
    var Name_abbr_ch =
      document.getElementById("name_abbr_ch").value;     //中文简称
    var Name_abbr_en =
      document.getElementById("name_abbr_en").value;     //英文简称
    var str = "CountryID1=" + CountryID1 + "&CountryID2="
      + CountryID2 + "&CountryID3=" + CountryID3
      + "&Name_ch=" + Name_ch + "&Name_en=" + Name_en
```

```
     + "&Name_abbr_ch=" + Name_abbr_ch
     + "&Name abbr en=" + Name abbr en;
   return str;     //返回要发送的数据参数字符串
}
```

回调函数 handleStateChange()的代码也很简单，直接将返回结果以 HTML 形式插入到 HTML 的位置：

```
function handleStateChange() {
   if (XmlHttp.readyState == 4) {
      //判断对象状态
      if (XmlHttp.status == 200) {
         document.getElementById("res").innerHTML = XmlHttp.responseText;
      }
   }
}
```

最后，来看看服务器端 sCountry.aspx 文件的内容，它接受了客户端 POST 过来的数据，进行处理后输出结果字符串：

```
protected void Page Load(object sender, EventArgs e)
{
   string CountryID1 = Request["CountryID1"];
   string CountryID2 = Request["CountryID2"];
   string CountryID3 = Request["CountryID3"];
   string Name ch = Request["Name ch"];
   string Name en = Request["Name en"];
   string Name abbr ch = Request["Name abbr ch"];
   string Name abbr en = Request["Name abbr en"];
   Response.Write("<h3>提交成功，请确认所填写的国家代码表信息。</h3>");
   Response.Write("<ul>");
   Response.Write("<li>三位数字代码：" + CountryID1 + "</li>");
   Response.Write("<li>两位英文字母代码：" + CountryID2 + "</li>");
   Response.Write("<li>三位英文字母代码：" + CountryID3 + "</li>");
   Response.Write("<li>中文名称：" + Name ch + "</li>");
   Response.Write("<li>中文简称：" + Name_abbr_ch + "</li>");
   Response.Write("<li>英文名称：" + Name en + "</li>");
   Response.Write("<li>英文简称：" + Name_abbr_en + "</li>");
   Response.Write("</ul>");
   Response.End();
}
```

看到了吧，无论是 GET 还是 POST 方式，其实在 ASP.NET 中都可以直接用 Request 对象来取值。是不是感觉很强大啊。

7.6.4　运行结果

执行 HTML 文件，让我们一起来验证奇迹的发生。出现页面后，在表单中输入数据，接

着单击“提交”按钮，看看发了什么？

在 check.aspx 中非常正确地显示了所填写的信息，并要求用户进行确认，效果如图 7-13 所示。



图 7-13　添加国家代码表效果

7.6.5　实例分析



源码解析：

整个实例的运行流程与前面基本一致。createDataString()函数则是实例的重点，它将用户在表单中输入的内容转换成将要以 POST 方式发送到 Ajax 服务器端的数据。

该函数的返回值将作为 XMLHttpRequest 对象 send()方法的参数提交到服务器端的 sCountry.aspx 文件。接着是 XMLHttpRequest 对象的回调函数，根据输出的内容直接在 HTML 中显示。

7.7　服务器端 Ajax 实现

从 Ajax 诞生以来，人们就发现使用 Ajax 框架可以带来极大的方便，节省了大量的时间和精力。于是，出现了大量的 Ajax 框架，有些框架基于客户端，有些基于服务器端。

据说，微软这个软件家族看到 Ajax 这么火，也坐不住了。连夜召开大会，命令研发小组准备将它收入囊中，并在内部称开发代号为 Atlas。

同时，微软的公关向外界发布新闻，称 Atlas 是多么的神奇，是多么伟大的创新，功能多么的好。

终于，在 2007 年初推出了其第一个正式版本，并将 Atlas 更名为 ASP.NET Ajax。它实质上是一个服务器端的 Ajax 框架，包括 Ajax 功能扩展以及 Ajax 服务器端控件集两部分。最初的 ASP.NET Ajax 1.0 在 ASP.NET 2.0 之上以一个单独安装的形式发布，供用户下载使用。

从.NET 3.5 开始，所有这些特性都成为 ASP.NET 所内置的。这意味着在构建或部署应用时，不再需要额外下载和安装单独的 ASP.NET Ajax 安装文件。只需要在 Visual Studio 2008 中创建针对.NET 3.5 的 ASP.NET 应用程序或网站项目，Visual Studio 2008 会自动在 Web.config 文件里添加适当的 Ajax 注册设置，而且核心 ASP.NET Ajax 服务器控件会出现在工具箱里，如图 7-14 所示。

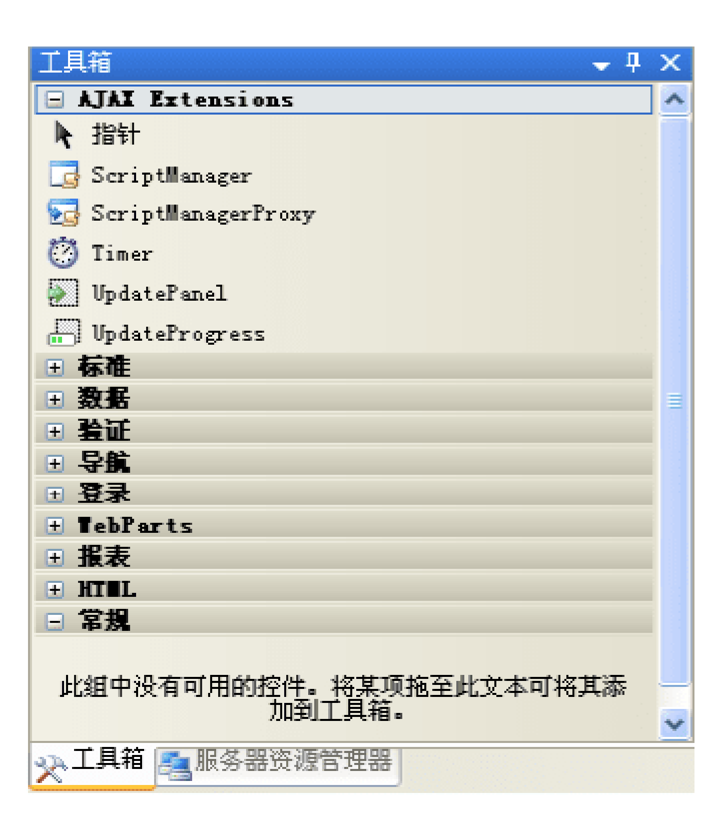


图 7-14　Ajax 工具箱

7.7.1　动态添加新闻效果

在 ASP.NET Ajax 这个大家庭中主要包括 5 个主角，每个都有着高超的本领。它们分别是 ScriptManager、ScriptManagerProxy、Timer、UpdatePanel 和 UpdateProgress，在本章后面将会对它们进行详细介绍。

 **视频教学：** 光盘/videos/7/ScriptManager.avi　　长度：5 分钟

1．基础知识——ScriptManager 控件

在一个 ASP.NET 页面中只能包含一个 ScriptManager 控件，且它必须出现在任何 Ajax 控件之前。因此，在 ASP.NET Ajax 开发中 ScriptManager 控件很重要，也是 ASP.NET Ajax 的核心组成之一。

可以这样理解，你若想在 ASP.NET 中使用 Ajax 功能，就必须先向老大 ScriptManager 进行报到。

下面就来看看老大的实力如何，具体哪些工作是它帮我们做的。

其中 CompositeScript 和 RoleService 模板是 ASP.NET 3.5 中新增的：

```
<asp:ScriptManager ID="ScriptManager1" runat="server" >
    <AuthenticationService Path="" />
    <ProfileService LoadProperties="" Path="" />
    <Scripts>
        <asp:ScriptReference />
    </Scripts>
    <Services>
        <asp:ServiceReference />
    </Services>
    <CompositeScript Path="" ScriptMode="Auto"/>
    <RoleService  LoadRoles="false" Path=""/>
</asp:ScriptManager>
```

接下来，我们来了解一下该控件提供的重要属性，如表 7-4 所示。只有这样用户才能更好地使用该控件来开发出符合要求的项目。

表 7-4　ScriptManager 的属性

属　性	说　明
AsyncPostBackErrorMessage	异步回传发生错误时的自定义提示错误信息
AsyncPostBackTimeout	异步回传时超时限制，默认值为 90，单位为秒
EnablePageMethods	表示是否使用当前页的静态方法
EnableScriptLocalization	表示是否启用脚本的本地化功能
ScriptMode	指定 ScriptManager 发送到客户端的脚本的模式，有 4 种模式：Auto、Inherit、Debug 和 Release，默认值为 Auto
Scripts	页面所有的脚本集合
Services	页面相关的 Web Service

2. 基础知识——引用 UpdatePanel 控件

UpdatePanel 控件是 ASP.NET Ajax 当中的一个重要成员，它与 ScriptManager 配合之后，可以通过相当简单的方式实现页面异步局部更新的效果。它必须依赖于 ScriptManager 存在，因为 ScriptManger 控件提供了客户端脚本生成与管理 UpdatePanel 的功能。

为此，需要把 UpdatePanel 控件加到页面中，然后在<ContentTemplate></ContentTemplate>中加入想要局部更新的内容。UpdatePanel 控件的简单定义形式如下：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <!—此处加入局部更新的内容-->
    </ContentTemplate>
</asp:UpdatePanel>
```

根据 UpdatePanel 控件的 RenderMode 属性值，<ContentTemplate>模板的内容显示在<div>或者<span>元素中。这个属性的默认值是 Block，即显示在<div>元素中，要使用<span>元素，

应该将 RenderMode 属性设置为 Inline。

3．实例描述

某巨星要来开演唱会啦，某商场又有新特价活动啦，公司楼下新开张了一家饭馆，菜非常不错还很实惠哦，中午我们一起去尝尝吧。

早已习惯了每天早上同事们的“新闻天下”，什么八卦新闻、小道消息和衣食住行，可谓是什么都有。呵呵，丰富了我们的早餐生活，还增长了见识。

话不多说了，转入正题。今天将继续 ASP.NET Ajax 的旅程，首先出场的是 ScriptManager，以及它最亲密的伙伴 UpdatePanel。

下面就来看看它们是如何实现动态添加新闻效果的吧，响应速度非常快，几乎感觉不到页面在刷新。

4．实例应用

【例 7-7】动态添加新闻效果。

首先，将 ScriptManager 老大从 ASP.NET Ajax 工具箱中请出来，放置在页面中坐观其变(当大哥就是好，有吃有喝，还啥事也不做)：

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

大哥来了，弟兄们也可以开工了。

既然是添加新闻，那至少得包含一个用于输入标题的文本框和一个添加按钮。

要实现动态添加的效果，我们还得再请一位，是这方面的专家，只需将我们的东西和要求给它。剩下的时间就可以睡觉去了，非常省心又快捷。

这位神人就是 UpdatePanel 老兄。如下所示即为它来之后所做的安排：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        <div class="title">News | 企业新闻</div>
        <div class="body">
            <p>
            标题：<asp:TextBox ID="txtNewsTitle" runat="server" Height="16px"
              Width="310px"></asp:TextBox>
            <asp:Button ID="btnAddNews" runat="server" Text="添加"
              OnClick="btnAddNews_Click" /><br />
            </p>
            <asp:BulletedList ID="NewsList1" runat="server">
            </asp:BulletedList>
        </div>
    </ContentTemplate>
</asp:UpdatePanel>
```

上面这层关系处理好之后，事还没有完，还得提出业务的逻辑顺序供它参考。

在本例中分为两步：先获取输入值再添加到列表中，代码如下：

```
protected void btnAddNews_Click(object sender, EventArgs e)
```

```
{
    string strTitle = txtNewsTitle.Text;
    NewsList1.Items.Add(strTitle + "    " + DateTime.Now.ToString());
}
```

5. 运行结果

本例中为了区分新闻添加的次序，在标题之后添加了时间。运行实例，在“标题”文本框中输入一些内容，再单击“添加”按钮查看结果。

此时页面没有刷新，而是直接添加到标题列表中，这说明添加的 UpdatePanel 控件在起作用。如图 7-15 所示是添加多个标题之后的效果。

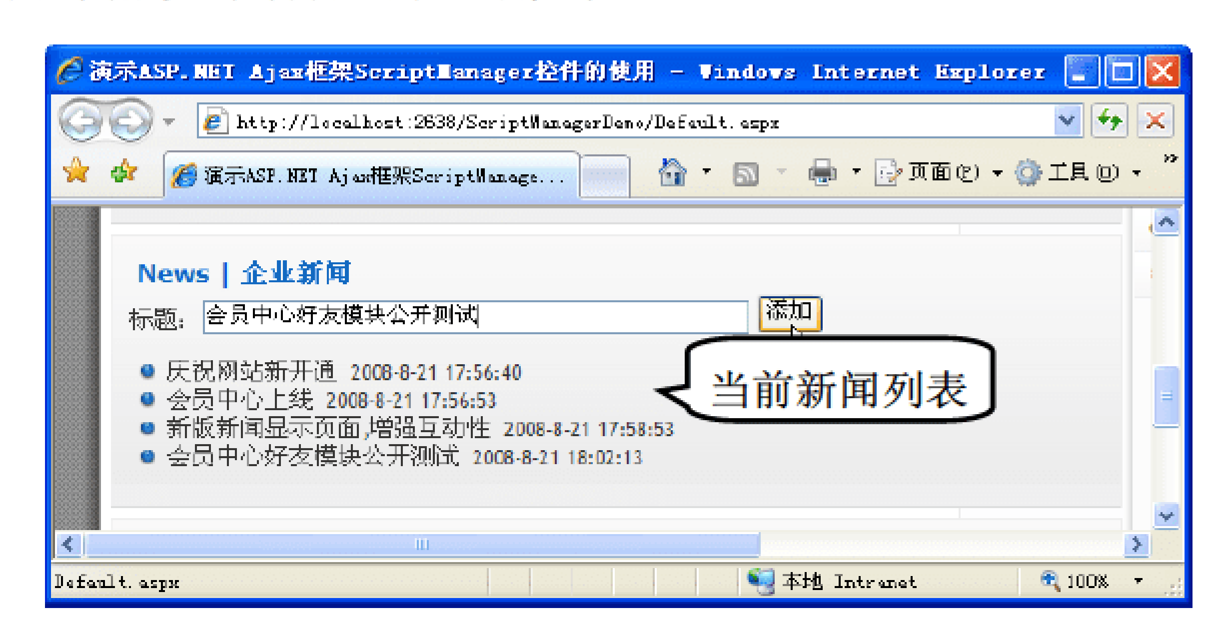


图 7-15　动态添加新闻效果

6. 实例分析



源码解析：

实例中不需要过多地编写后台代码，也不用去繁琐地编写 JavaScript 代码。所需的工作，仅仅是添加 ScriptManager 和 UpdatePanel，之后页面就发生了神奇的变化。

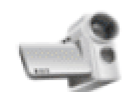
但是要注意 ScriptManager 控件必须位于页面最顶部，在任何控件之前，一般将它放在 <FORM>标记下方。对于 UpdatePanel 控件，直接将布局代码嵌入它的 Template 模板内即可。这样，在其中包含的内容在产生页面回调时不会刷新整个页面。

7.7.2　网站登录模块

登录模块能够配合注册模块让网站应用能够同用户进行信息交互，当用户在网站进行注册后，就需要登录模块进行用户登录。

登录模块虽然看上去比较容易，但是它与其他模块的关系非常密切，处理也比注册模块要复杂一些。

下面我们来学习在 ASP.NET 的用户控件中如何使用 Ajax 实现会员登录。



视频教学：光盘/videos/7/ScriptManagerProxy.avi　　长度：3 分钟

1. 基础知识——ScriptManagerProxy 控件

ScriptManager 是管理客户端脚本资源的，在页面中随时可以用。但是它有个限制，就是一个页面中只能有一个，所以当母版页里面有个 ScriptManager 的时候，内容页的 aspx 里面就不能再有了。

这时候，如果内容页里面要引用 ScriptManager 的话，那就可以放一个 ScriptManagerProxy，那么就可以通过它访问母版页的 ScriptManager 了。从字面也可以理解。proxy 是代理的意思，意为它是访问 ScriptManager 的代理，相当于代理服务器。

由于仅仅是代理，因此在 ScriptManagerProxy 控件中只能添加新的脚本或 WebService，而不能移除 ScriptManager 控件中已经添加了的脚本或 WebService。

2. 实例描述

一个巴掌拍不响，众人拾柴火焰高……这些都说明了多人协作或者团队的重要性。

开发一个软件更是如此。合理分工，才能充分发挥每个成员的优势，达到效率最优化。

就拿最近的一个 IDC 项目来说吧，其中很多功能在我们之前都有过类似的。唯一不同的是对会员这一块的需求比较多。因此，在开发时就调用了我们最精锐的开发人员在这里，其他人员则按需在原有模块基础上进行调整。为了突出 IDC 行业即时的特点，在开发时选择了最流行的 ASP.NET Ajax 技术，并采用用户控件来封装。

3. 实例应用

【例 7-8】网站登录模块。

首先创建一个用户控件 WebUserControl.ascx，在该控件内进行编辑，创建一个会员登录表单，代码如下：

```
<%@ Control Language="C#" AutoEventWireup="true"
  CodeBehind="UserControl.ascx.cs" Inherits="DomainService.UserControl" %>
    <div class="title">
    Member Center | 会员中心</div>
<a href="#">
    UserName:
    <asp:TextBox ID="txtname" runat="server"></asp:TextBox><br />
    PassWord:
    <asp:TextBox ID="txtpwd" runat="server"
      TextMode="Password"></asp:TextBox><br />
    <asp:Button ID="Button1" runat="server" OnClick="Button1 Click" Text="登录" />
</a>
```

由于用户控件要被应用到页面中才有效，并且用户控件也要使用 Ajax，而一个页面上只能有一个 ScriptManager 控件。因此，这步要求在用户控件中添加 ScriptMangerProxy 控件来代替 ScriptManger 控件，这里还引用了一个 Web Service，代码如下：

```
<asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server">
    <Services>
        <asp:ServiceReference Path="~/WebService1.asmx" />
    </Services>
</asp:ScriptManagerProxy>
```

为用户控件的.cs 文件添加“登录”按钮的单击代码实现登录判断，代码如下：

```
protected void Button1_Click(object sender, EventArgs e)
{
    WebService1 checkWS = new WebService1();
    if (checkWS.IsLoginSuccess(txtname.Text, txtpwd.Text))
    {
        Page.ClientScript.RegisterStartupScript(GetType(), "js",
          "alert('登录成功，即将转到会员中心页面！');", true);
    }
    else
    {
        Page.ClientScript.RegisterStartupScript(GetType(), "js",
          "alert('登录失败，请重试或使用找回密码！');", true);
    }
}
```

由于 ScriptManager 与 ScriptManagerProxy 控件很相似，所以 ScriptManagerProxy 控件同样也可以在客户端和服务器端调用。上述的单击代码中，调用了 Web Service 的 IsLoginSucess() 方法，该方法返回一个逻辑值判断是否登录成功。

保存上述的操作。将用户控件 WebUserControl.asx 文件拖到要显示登录的位置，再切换到“代码”视图，在页面顶部会看到自动生成的注册指令：

```
<%@ Register src="WebUserControl.ascx" tagname="WebUserControl"
  tagprefix="uc1" %>
```

同时在显示用户控件的位置也使用了特殊的引用指令：

```
<uc1:WebUserControl ID="WebUserControl1" runat="server" />
```

4. 运行结果

最后，看一下实例的运行效果。如图 7-16 所示为单击“登录”按钮后登录失败时的效果。

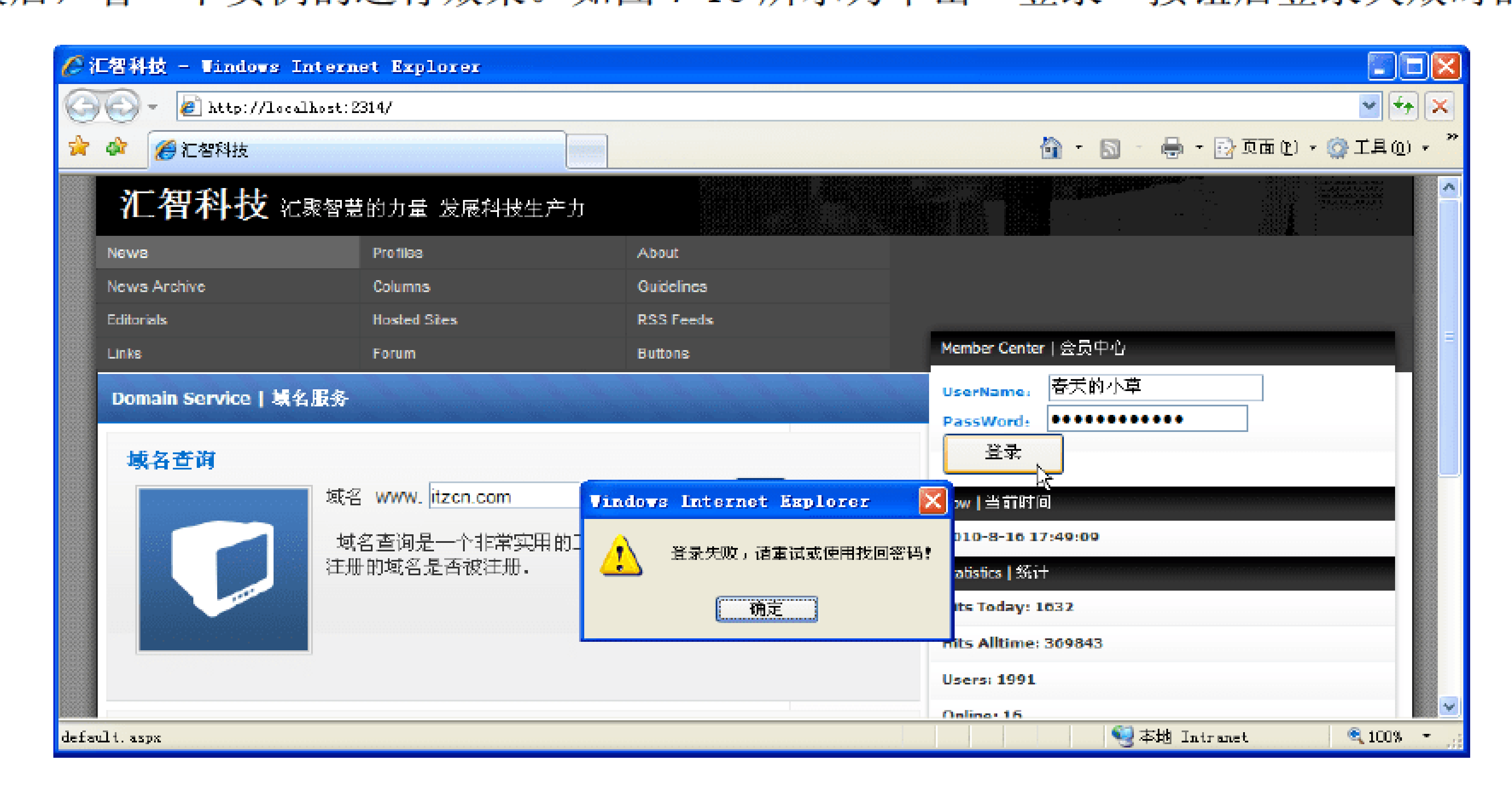


图 7-16 登录模块的运行效果

5. 实例分析


源码解析：

用户控件的最大特点就是可以在页面中多次引用，所以选择它来做登录模块。另外，为了保证登录模块与页面中其他模块不相冲突，必须使用 ScriptManagerProxy 控件来化解。

还通过 ScriptManagerProxy 控件的 Services 模板引用了一个 Web Service，也从最大程度上保证了模块之间的松散耦合。

7.7.3　异步更新的日期选择界面

在 ASP.NET Ajax 的框架中，UpdateProgress 控件是 UpdatePanel 控件的“兄弟”，它的使用方法很简单。作用就是在 UpdatePanel 控件工作时提示用户正在处理，这在处理大数据量或者复杂的业务逻辑时非常有用。

本节将通过一个日历控件来讲解这两个“兄弟”是如何协作的。


视频教学：光盘/videos/7/UpdateProgress.avi　　　　长度：7 分钟

1. 基础知识——UpdateProgress 控件

UpdateProgress 控件是 ASP.NET Ajax 框架中最简单的控件，用于当页面异步更新数据时显示给用户友好的提示信息。该信息可以是文本信息，也可以是图片信息，用户可以根据自己的项目需要进行选择。UpdateProgress 控件的定义形式如下：

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
   <ProgressTemplate>
      <!--表示进度的信息-->
   </ProgressTemplate>
</asp:UpdateProgress>
```

在默认情况下，UpdageProgress 控件将显示页面上所有 UpdatePanel 控件更新的进度信息。在以前版本的 UpdateProgress 中，无法设置让 UpdateProgress 只显示某一个 UpdatePanel 的更新。而在最新版本的 UpdateProgress 控件中提供了 AssociatedUpdatePanelID 属性，可以指定 UpdateProgress 控件显示哪一个 UpdatePanel 控件。

下面给出 UpdateProgress 控件的 3 个重要属性。

- AssociatedUpdatePannelID：通常用于有多个 UpdatePanel 的情况下，设置为与 Update Progress 相关联 UpdatePanel 的 ID。
- DisplayAfter：进度信息被展示后的 ms 数。
- DynamicLayout：UpdateProgress 控件是否动态绘制，而不占用网页空间。

2. 基础知识——UpdatePanel 控件的用法

在一个页面中没有限制使用 UpdatePanel 控件的数量，也就是说可以不使用、使用一个或多个。所以可以为不同的需要局部更新的页面区域加上不同的 UpdatePanel 控件。如图 7-17 所

示了在页面上存在多个 UpdatePanel 控件的情况。此时，更新一个 UpdatePanel 控件 3 并不会连带更新 UpdatePanel 控件 1。但当其嵌套使用时，最外面的 UpdatePanel 控件 1 被触发更新，里面的 UpdatePanel 控件 2 也随着更新。而当里面的 UpdatePanel 控件 2 触发更新时，并不会更新外层的 UpdatePanel 控件 1。

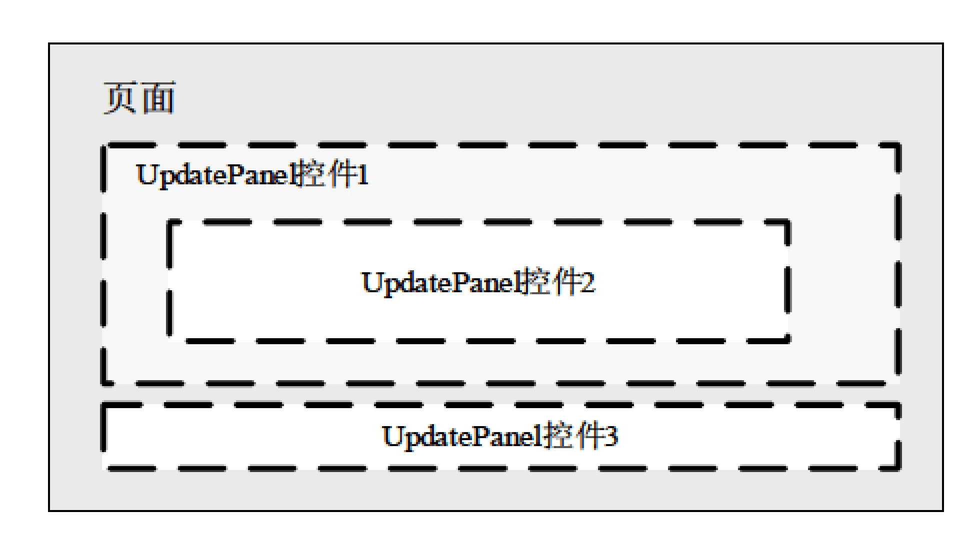


图 7-17　多个 UpdatePanel 控件共存

UpdatePanel 控件嵌套的简单定义代码如下：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
   <ContentTemplate>
      <!--顶层内容-->
      <asp:UpdatePanel ID="UpdatePanel2" runat="server">
         <ContentTemplate>
            <!--嵌套内容-->
         </ContentTemplate>
         <Triggers>
         </Triggers>
      </asp:UpdatePanel>
   </ContentTemplate>
   <Triggers>
   </Triggers>
</asp:UpdatePanel>
```

UpdatePanel 控件作为 ASP.NET Ajax 中实现页面局部刷新的重要控件，它也有一些自己的属性，如下所示。

- ChildrenAsTriggers：应用于 UpdateMode 属性为 Conditional 时，指定 UpdatePanel 中的子控件的异步回送是否会引发 UpdatePanle 的更新。
- RenderMode：表示 UpdatePanel 最终呈现的 HTML 元素，默认 Block 表示<div>，Inline 表示<span>。
- Triggers：用于引起更新的事件。
- UpdateMode：表示 UpdatePanel 的更新模式，可以是 Always 或者 Conditional。



Always 表示无论有没有 Trigger，其他控件都将触发 UpdatePanel。Conditional 表示只有当前 UpdatePanel 的 Trigger 或 ChildrenAsTriggers 属性为 true 时，才会引发更新 UpdatePanel。

为了避免页面上一个局部更新被触发则所有的 UpdatePanel 都将更新这种情况，需要把

UpdateMode 属性设置为 Conditional，然后为每个 UpdatePanel 设置专用的触发器。

下面的代码表示了两个共存的 UpdatePanel 控件：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="always">
    <ContentTemplate>
        <%= DateTime.Now %>
        <asp:Button ID="Button1" runat="server" Text="更新 1" />
    </ContentTemplate>
</asp:UpdatePanel>
<hr />
<asp:UpdatePanel ID="UpdatePanel2" runat="server" UpdateMode="Conditional"
  ChildrenAsTriggers="false">
    <ContentTemplate>
        <%= DateTime.Now %>
        <asp:Button ID="Button2" runat="server" Text="更新 2" />
    </ContentTemplate>
</asp:UpdatePanel>
```

在上述代码所示的页面中，如果用户单击“更新 1”按钮，将更新自己的 UpdatePanel1，而不能更新 UpdatePanel2，因为 UpdatePanel2 的 UpdateMode 属性值为 Conditional。

如果单击“更新 2”按钮，将会更新 UpdatePanel1，因为在 UpdatePanel2 中设置了 ChildrenAsTriggers 属性值为 false，所以不能更新自己。希望读者能自己尝试设置这几个属性的值，通过不同的执行结果体会具体的含义。

3. 实例描述

最近做了一个关于会员模块的作品，此作品以无刷新的异步通信为主，包含会员登录、会员修改个人资料等。例如，在修改个人的出生日期时，如果网络繁忙会提示正在处理请稍候，整个界面和效果非常友好。

下面，就将此方法公布出来，希望对正在研究的读者有所帮助。

4. 实例应用

【例 7-9】异步更新的日期选择界面。

(1) 要用 ASP.NET Ajax 实现，第 1 步当然是创建一个 ASP.NET 页面，再把 ScriptManager 控件放到页面上。

(2) 紧接着我们把 UpdatePanel 控件也添加进来，后面的操作都是在它的 ContentTemplate 模板内完成的。

(3) 在页面要显示选择日期的位置，添加两个 DropDownList 控件。一个用于选择月份的 DropDownList1，另一个用于选择年份的 DropDownList2。具体的布局代码就不再给出了，比较简单，但要注意的是需要设置 AutoPostBack 属性为 True。

(4) 现在隆重介绍日期选择控件——Calendar，它可以提供一个选择日期的月历。而且该控件还内置了很多种颜色格式，这里选择了彩色 2 型。如下是生成的代码：

```
<asp:Calendar ID="Calendar1" runat="server" BackColor="White"
  BorderColor="#3366CC" BorderWidth="1px" DayNameFormat="Shortest"
  Font-Names="Verdana" Font-Size="8pt"
```

```
  ForeColor="#003399" Height="200px" Width="220px"
  OnSelectionChanged="Calendar1_SelectionChanged"
  SelectedDate="2008-03-29" CellPadding="1">
<SelectedDayStyle BackColor="#009999" Font-Bold="True" ForeColor="#CCFF99" />
<WeekendDayStyle BackColor="#CCCCFF" />
<TodayDayStyle BackColor="#99CCCC" ForeColor="White" />
<SelectorStyle BackColor="#99CCCC" ForeColor="#336666" />
<OtherMonthDayStyle ForeColor="#999999" />
<NextPrevStyle Font-Size="8pt" ForeColor="#CCCCFF" />
<DayHeaderStyle BackColor="#99CCCC" Height="1px" ForeColor="#336666" />
<TitleStyle BackColor="#003399" Font-Bold="True" Font-Size="10pt"
  ForeColor="#CCCCFF"
  BorderColor="#3366CC" BorderWidth="1px" Height="25px" />
</asp:Calendar>
```

(5) 从 Calendar 控件下方安排一个位置，添加一个 Label 控件，这个控件用于显示在日历中被选择的日期，以及当前的日期。

(6) 在页面布局完成后，下面就要为该页面编写后台代码了，双击 Calendar1 控件的 SelectionChanged 事件。该事件在用户更改选择日期时激发，在进入的后台代码中编写获取所选择的日期及当前时间的代码，具体代码如下：

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(4000);
    Label1.Visible = true;
    Label1.Text = "你选定日期是：" + Calendar1.SelectedDate.ToLongDateString();
    Label1.Text =
      Label1.Text + "<br/>今天日期是：" + DateTime.Now.ToLongDateString();
}
```

(7) 依次进入 DropDownList1 和 DropDownList2 控件的 SelectedIndexChanged 事件，在这里编写代码实现选择之后更新到日历控件 Calendar1。

(8) 通过“属性”窗口打开 UpdatePanel1，设置 UpdateMode 属性为 Conditional。

(9) 再找到 Trigger 属性，在这里编辑都有哪些事件可以引起控件本身的更新。例如，添加 DropDownList1 成员 SelectedIndexChanged 行为，效果如图 7-18 所示。

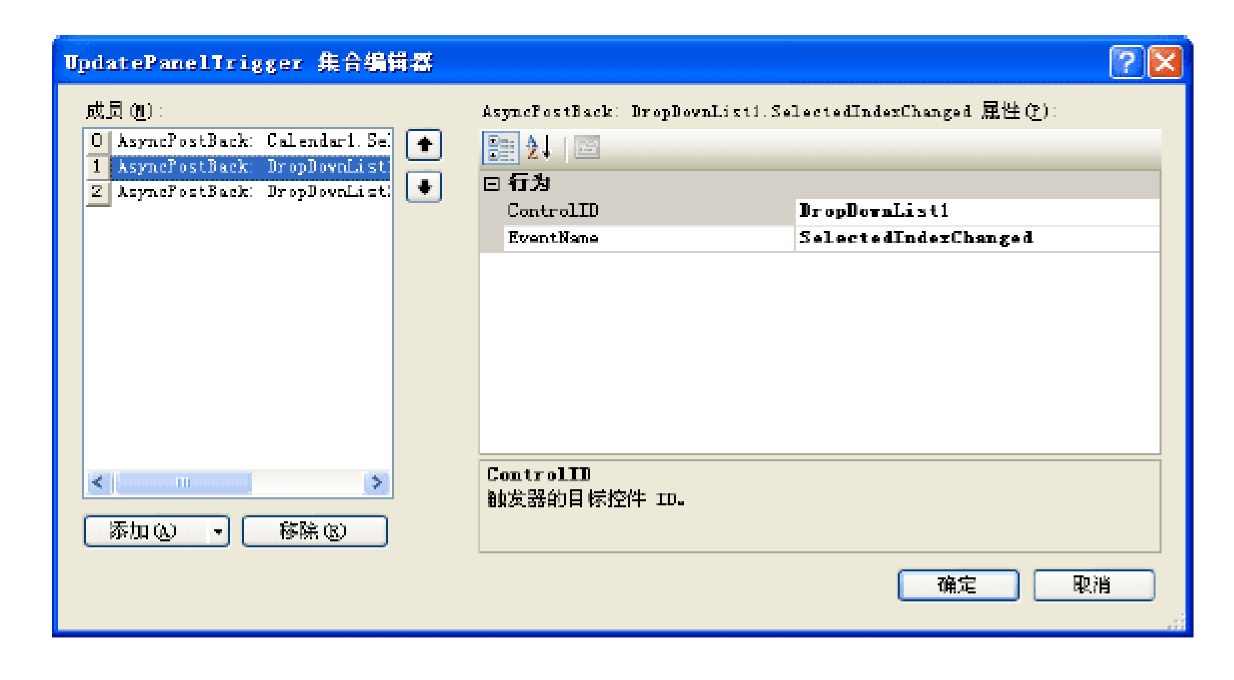


图 7-18　添加 Triggers 属性的成员和行为

(10) 在 Triggers 属性的成员和行为添加完成后，页面的 UpdatePanel 控件源文件中将添加相应的代码，代码如下所示：

```
<asp:UpdatePanel>
    <Triggers>
    <asp:AsyncPostBackTrigger ControlID="Calendar1"
      EventName="SelectionChanged" />
    <asp:AsyncPostBackTrigger ControlID="DropDownList1"
      EventName="SelectedIndexChanged" />
    <asp:AsyncPostBackTrigger ControlID="DropDownList2"
      EventName="SelectedIndexChanged" />
    </Triggers>
</asp:UpdatePanel>
```

(11) 最后，添加一个 UpdateProgress 控件使处理过程中能显示一个加载图片和文字。如下所示为使用后的代码，它同样位于 UpdatePanel 控件的 ContentTemplate 模板内：

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server">
    <ProgressTemplate>
        <div style="color: #FF0000">
            <img src="loader.gif" />正在处理，请稍候....
        </div>
    </ProgressTemplate>
</asp:UpdateProgress>
```

5. 运行结果

至此制作完成，运行页面查看效果。首先，在年份和月份中进行选择，可以看到下方将会实时显示当前的月历，很明显 UpdatePanel 控件已经起了作用。

例如，选择 1985 年 12 月 18 日这一天并在月历中单击，此时效果如图 7-19 所示。



图 7-19　选择日期后的处理效果

此时，将会看到在 UpdateProgress 控件内设置的加载动画和提示文本。待处理完后，将显示选择的日期和今天的日期，如图 7-20 所示。

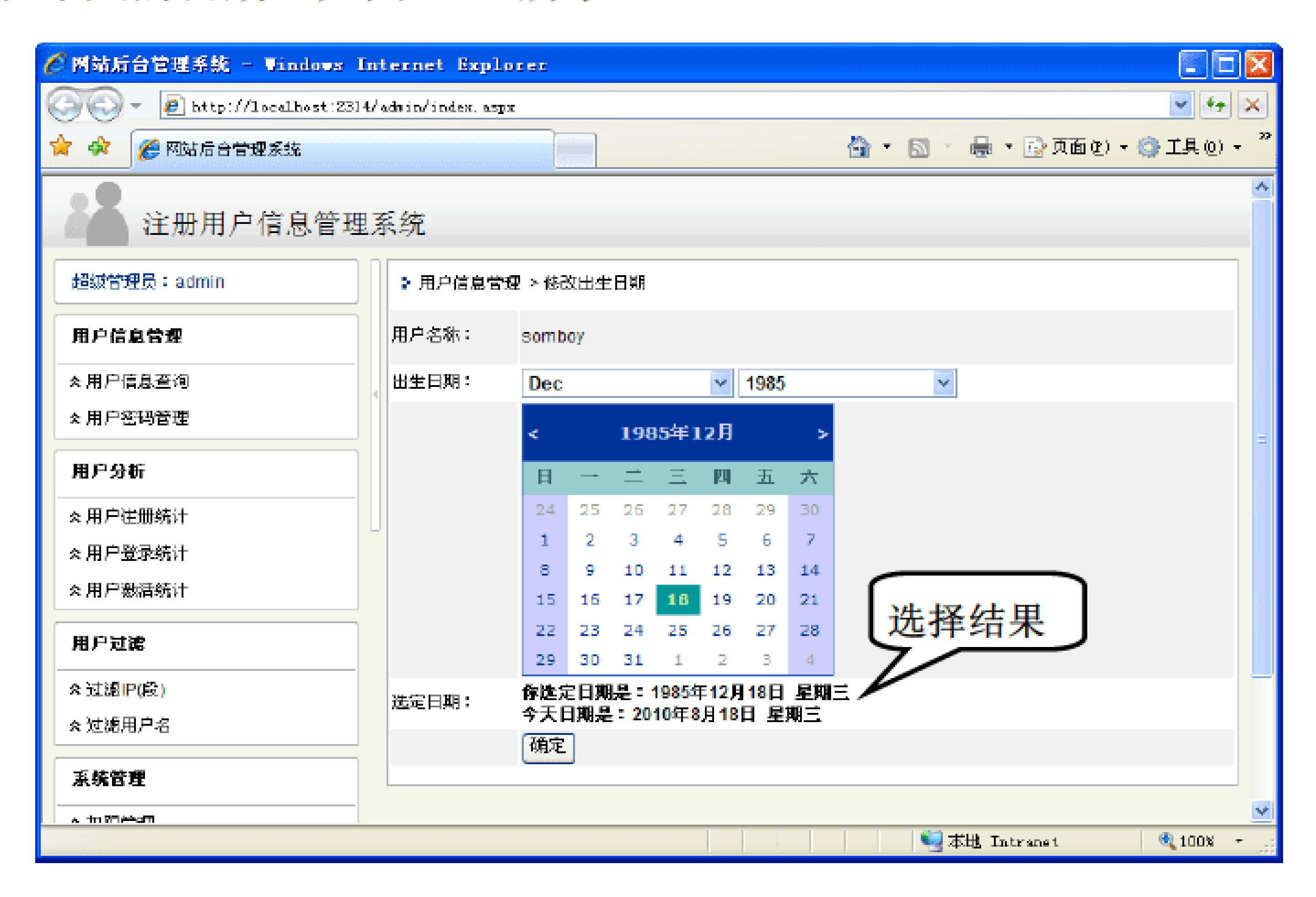


图 7-20　显示选择的日期

6. 实例分析



源码解析：

由运行效果可以看到，在添加 UpdateProgress 控件之后，无须做任何其他的设置，即可正常运作。

另外，由于在 UpdatePanel 中设置了 UpdateMode 属性为 Conditional，使得只有在 Triggers 集合中指定的控件和事件才会触发 UpdatePanel 控件的无刷新。

Triggers 集合还支持另一种称为同步触发器(PostBackTrigger)的方式，它采用传统的整页传输机制。读者可以试试改用这种方式后的效果。

7.7.4　实现监视服务器性能计数器

在 Web 应用程序中常常要用到时间控制的功能，如在程序界面上显示当前时间，或者每隔多长时间触发一个事件等。为此，微软在 ASP.NET Ajax 中为我们提供了 Timer 控件，它可以轻松地实现这样的功能。



视频教学： 光盘/videos/7/Timer.avi　　　　**长度：** 5 分钟

1. 基础知识——Timer 控件

在 Web 应用程序中常常要用到时间控制的功能，如在程序界面上显示当前时间，或者每隔多长时间触发一个事件等。为此在 ASP.NET Ajax 中提供了 Timer 控件，Timer 控件负责以固定的时间间隔向服务器发送同步或异步的请求。

Timer 控件经常和 UpdatePanel 控件结合起来使用以实现定时异步更新页面一部分的功能。另外，也可以使用这个控件定期 PostBack 整个页面。

Timer 控件的使用分两种情况，一种是位于 UpdatePanel 控件的内部，具体格式如下面的代码所示：

```
<asp:ScriptManager ID="ScriptManager2" runat="server" ></asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
   <ContentTemplate>
      <asp:Timer ID="Timer1" runat="server" Interval="10000"
        OnTick="Timer1 Tick">
      </asp:Timer>
   </ContentTemplate>
</asp:UpdatePanel>
```

另一种是 Timer 控件位于 UpdatePanel 控件的外部，具体格式如下面的代码所示：

```
<asp:ScriptManager ID="ScriptManager1" runat="server" ></asp:ScriptManager>
<asp:Timer ID="Timer1" runat="server" Interval="10000" OnTick="Timer1 Tick">
</asp:Timer>
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Conditional">
   <Triggers>
      <asp:AsyncPostBackTrigger ControlID="Timer1" EventName="Tick" />
   </Triggers>
   <ContentTemplate>
   <!--显示内容-->
   </ContentTemplate>
</asp:UpdatePanel>
```

这里需要说明的是，虽然 Timer 控件是一个服务器控件，而事实上它会在页面内嵌入一个 JavaScript 组件。此 JavaScript 组件会在 Timer 控件的 Interval 属性所设置的间隔时间到达时从浏览器引发回送。用户可以利用服务器端程序代码来设置 Timer 控件的属性，而所做的设置会传递给 JavaScript 组件。

Timer 控件有两个个重要的属性及一个常用事件。

- Enabled 属性：用于表示是否 tick 事件。
- Interval 属性：用于指定间隔时间。
- Tick 事件：指定间隔到期后执行。

技术文档	Timer 控件位置不同及其区别

Timer 控件在 UpdatePanel 控件的内外是有区别的。当 Timer 控件在 UpdatePanel 控件内部时，JavaScript 计时组件只有在一次回传完成时才会重新建立。也就说直到网页回传之前，定时器间隔时间不会从头计算。

例如，用户设置 Timer 控件的 Interval 属性值为 6000ms(6s)，但是回传操作本身却花了 2s 才完成，则下一次的回传将发生在前一次回传被引发之后的 8s。而如果 Timer 控件位于 UpdatePanel 控件之外，则当回传正在处理时，下一次的回传仍将发在前一次回传被引发之后的 6s。也就是说，UpdatePanel 控件的内容被更新之后的 4s，就会再次看到该控件被更新。

2. 实例描述

Ajax 不仅能在客户端提高程序的可操作性，而且同样能作用于服务器端。特别是像天气预报、在线人数统计以及股票趋势图等这些对信息要求及时的场合。

这里我们的实例将利用 Ajax 技术监视服务器上各个性能计数器的状态，实时反映运行情况，从而为管理员提供一个参考。

3. 实例应用

【例 7-10】实现监视服务器性能计数器。

(1) 打开网站后台的系统页面，将 ScriptManager 控件添加到布局上。

(2) 添加一个 UpdatePanel 控件。在其中使用 DropDownList 控件来制作一个更新时间的间隔列表，最终代码如下：

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
    <ContentTemplate>
        请选择服务器之性能技术器的更新时间间隔：
        <asp:DropDownList ID="DropDownList1" runat="server"
          AutoPostBack="True" Width="128px"
          OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
            <asp:ListItem Value="1000" Selected="True">1 秒钟</asp:ListItem>
            <asp:ListItem Value="3000">3 秒钟</asp:ListItem>
            <asp:ListItem Value="5000">5 秒钟</asp:ListItem>
            <asp:ListItem Value="10000">10 秒钟</asp:ListItem>
            <asp:ListItem Value="60000">1 分钟</asp:ListItem>
            <asp:ListItem Value="0">不更新</asp:ListItem>
        </asp:DropDownList>
    </ContentTemplate>
</asp:UpdatePanel>
```

(3) 在 UpdatePanel 控件之外添加一个 Timer 控件，用来定时更新：

```
<asp:Timer ID="Timer1" runat="server" OnTick="Timer1_Tick">
</asp:Timer>
```

(4) 设置一个 ID 为 lblPerformanceCounter 的 Label 控件来显示当前服务器的性能状态。

(5) 接下来切换到后台代码文件，在定时器的间隔事件中编写代码：

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    lblPerformanceCounter.Text = GetPerformanceCounter();   //更新显示
}
```

(6) 在 Page_Load 中添加更新显示的代码。当从列表中选择一下间隔时间后，OnSelectedIndexChanged 事件执行，它会使用指定间隔来更新状态。实现代码如下：

```
protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
    if (Convert.ToInt16(DropDownList1.SelectedValue) == 0)   //判断是否选择更新
    {
        Timer1.Enabled = false;                           //禁用定时器
```

```
    }
    else {                                               //更新显示
        Timer1.Interval = Convert.ToInt16(DropDownList1.SelectedValue);
        Timer1.Enabled = true;                           //启用定时器
    }
}
```

(7) GetPerformanceCounter()方法的主要内容为获取当前服务器的状态并返回一个字符串，具体的代码这里就不再给出，读者可以参考提供的源代码。

4. 运行结果

现在运行我们的程序，进入后台的系统页面。可以看到列出了服务器上的状态，如果在列表中更新了间隔，会在时间到期后更新，如图 7-21 所示。



图 7-21　监视服务器性能状态的运行效果

5. 实例分析


源码解析：

Timer 控件的属性和事件虽然不多，但要想掌握该控件的实际应用却并不容易。实例中将用户在 DropDownList 列表选中的间隔值赋予 Timer 控件的 Interval 属性，它的单位是毫秒。

当停留指定的间隔后触发 Tick 事件，一般在这里编写定时处理的代码，在本实例中为获取最新的服务器状态信息。

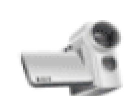
7.7.5　博客栏目分类

今天你开博了吗？

在 Internet 日益普及的今天，互联网上充斥着各种基于 Web 2.0 的网站，其中以博客(Blog)最为突出。因为博客注重的是用户之间的交流，而不限于内容和体裁。

作为一个有思想的网站开发员，时刻都在关注用户的需求，并努力提高自己的业务水平。

下面，我们就来看一下经常出现在博客中的表现形式是如何提高用户体验的。



视频教学：光盘/videos/7/Accordion.avi　　　　长度：12 分钟

1．基础知识——Accordion 控件

Accordion 是一个可以让你在 Web 页面方便地开发类似 Outlook 工具栏的 ASP.NET Ajax 控件，它可以使得你的页面用户能方便地展开或者关闭一系列页面元素的显示。

它有点类似多个 CollapsiblePanels 控件的组合。但是在一个时间内，它限制你的页面用户只能展开其中的一个部分，每一个 Accordion 控件包括若干个 AccordionPane 控件。

AccordionPane 控件可以像 Panel 控件一样，用来作为其显示内容的载体。

另外，每一个 AccordionPane 又具有 Header 和 Content 部分，分别用于表示它的标题和其中的内容。整个 Accordion 控件中各个子控件的层次结构如图 7-22 所示。

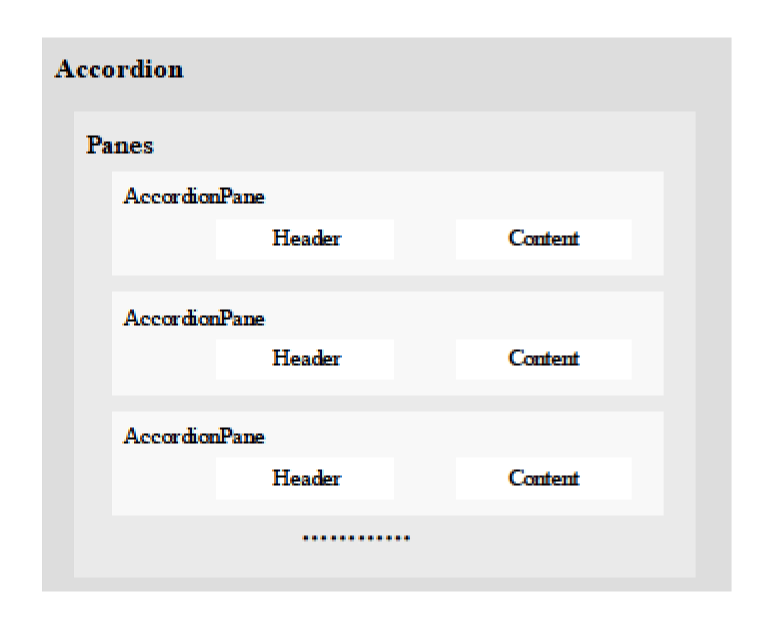


图 7-22　Accordion 控件的层次结构

由图 7-22 所示的结构映射到 Accordion 控件上，最终的语法类似如下所示：

```
<ajaxToolkit:Accordion ID="MyAccordion"
   runat="Server" SelectedIndex="0"
   HeaderCssClass="accordionHeader"
   HeaderSelectedCssClass="accordionHeaderSelected"
   ContentCssClass="accordionContent"
   AutoSize="None"  FadeTransitions="true"
   TransitionDuration="250"
   FramesPerSecond="40"
   RequireOpenedPane="false"
   SuppressHeaderPostbacks="true">
   <Panes>
      <ajaxToolkit:AccordionPane
         HeaderCssClass="accordionHeader"
         HeaderSelectedCssClass="accordionHeaderSelected"
         ContentCssClass="accordionContent">
         <Header> ... </Header>
         <Content> ... </Content>
      </ajaxToolkit:AccordionPane>
   </Panes>
   <HeaderTemplate>...</HeaderTemplate>
   <ContentTemplate>...</ContentTemplate>
</ajaxToolkit:Accordion>
```

表 7-5 中列出了 Accordion 控件常用的属性和事件。

表 7-5　Accordion 控件的常用属性

属　性	说　明
SelectedIndex	已显示的 AccordionPane 控件的索引号
HeaderCssClass	作用于标题的 CSS 样式
ContentCssClass	用于显示内容的 CSS 样式。使用类似于 HeaderCssClass
FadeTransitions	为 true 时为渐变效果，false 则为标准变换
TransitionDuration	在选择某一标题后伸展和收缩过渡动画所持续的时间，单位为毫秒
FramesPerSecond	用于伸展和收缩过渡动画每秒所需要的帧数
AutoSize	用于限制 Accordion 控件展开的高度，有 3 个可选值
Panes	AccordionPane 控件的集合
HeaderTemplate	标题模版，其中包含的标记用于进行数据绑定，来显示面板标题
ContentTemplate	内容模版，其中包含的标记用于进行数据绑定，来显示面板内容
DataSource	指定数据源，必须调用 DataBind()方法进行数据绑定
DataSourceID	用数据源的 ID 来指定一个数据源
DataMember	当使用 DataSourceID 来指定数据源时用于绑定成员

Accordion 控件具有保持其选中状态的功能，当页面发生提交的过程后，Accordion 保留其提交前选中的页面。

它支持以下 3 种显示和排版方式。

- None：Accordion 在其展开或者折叠过程中，将根据它内部显示的内容自动做尺寸的变化，不受到任何条件限制。如果将 AutoSize 属性设置为 None 它将可能造成页面上的其他元素跟随 Accordion 的尺寸变化，产生向上或者向下的移动。
- Limit：将使得 Accordion 控件永远不能将它的尺寸扩展到规定的高度(Height)属性之外，如果将 AutoSize 属性设置为 Limit，可能会造成在某种情况下，它里面的内容需要通过滚动条来滚动。
- Fill：将使得 Accordion 控件永远都保持在其高度(Height)属性规定的高度。

2. 基础知识——CollapsiblePanelExtender 控件

CollapsiblePanelExtender 控件是一个可以很方便地使用户折叠或者展开网页上任何一个部分的 ASP.NET Ajax 控件。开发人员需要做的只是指定一个页面上需要折叠展开功能的 Panel 为其 TargetID。

CollapsiblePanelExtender 控件的使用格式如下：

```
<ajaxToolkit:CollapsiblePanelExtender ID="cpe" runat="Server"
   TargetControlID="Panel1"
   CollapsedSize="0"
   ExpandedSize="300"
   Collapsed="True"
   ExpandControlID="LinkButton1"
   CollapseControlID="LinkButton1"
```

```
    AutoCollapse="False"
    AutoExpand="False"
    ScrollContents="True"
    TextLabelID="Label1"
    CollapsedText="Show Details..."
    ExpandedText="Hide Details"
    ImageControlID="Image1"
    ExpandedImage="~/images/collapse.jpg"
    CollapsedImage="~/images/expand.jpg"
    ExpandDirection="Vertical" >
<ajaxToolkit:CollapsiblePanelExtender>
```

CollapsiblePanelExtender 控件具有保持折叠和展开状态的功能，当网页被提交的时候，它将自动记录与恢复其目标控件的折叠或者展开状态。也可以指定当目标 Panel 展开的时候是否出现滚动条，并且设置该滚动条在高度或者宽度上的数据。

表 7-6 中给出了常用属性的说明。

表 7-6　CollapsiblePanel 控件的常用属性

属　性	说　明
TargetControlID	要实现折叠的 Panel 控件的 ID
CollapsedSize	用于指定折叠后的尺寸
ExpendedSize	用于指定伸展的尺寸
Collapsed	默认打开页面时，此 Panel 是否处于折叠状态
ExpandControlID	激发伸展效果的控件，主要是通过控件的 Click 事件实现伸展效果
CollapseControlID	激发折叠效果的控件，主要通过控件的 Click 事件实现折叠效果
AutoCollapse	当失去焦点时是否自动折叠
AutoExpand	当失去焦点是否自动伸展
ScrollContents	是否在 Panel 内显示滚动条
TextLabeID	显示折叠状态的目标控件
CollapsedText	折叠时显示的信息
ExpandedText	展开时显示的信息
ImageControlID	如果通过 Image 图像实现折叠和伸展效果，那么在此处设置图像的 ID
ExpandedImage	展开时使用的图像路径
CollapsedImage	折叠时使用的图像路径
ExpandDirection	伸展方面，水平伸展或垂直伸展

3．实例描述

通过上面的学习，我们对 Ajax 框架中的 CollapsiblePanelExtender 控件和 Accordion 控件有了很好的认识。有意思的是，这两个控件的网页效果十分类似，那么它们在实际运用的过程中有什么不同和值得注意的地方呢？

下面还是让我们通过实例，用 CollapsiblePanelExtender 控件实现“博客栏目分类栏”的收

缩效果，用 Accordion 控件实现“博客文章内容”的折叠效果，来体验一下它们的相同和不同之处吧！

4. 实例应用

【例 7-11】博客栏目分类。

先在页面上将 ScriptManager 控件和 UpdatePanel 控件放到 form1 表单里面，代码如下：

```
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
<asp:UpdatePanel ID="UpdatePanel1" runat="server">
<ContentTemplate> </ContentTemplate>
</asp:UpdatePanel>
</form>
```

在 UpdatePanel 控件中添加一个 Penal 控件，设定 ID 属性值为“PanelHeader1”。它是用于显示博客栏目的标题，具体代码如下：

```
<asp:Panel ID="PanelHeader1" runat="server">
    <h2><span><asp:ImageButton ID="PanelheaderImage1" runat="server"
     AlternateText="collapse"/>
     IT 技术杂谈</span></h2>
</asp:Panel>
```

接下来，再拖一个 Penal 控件到 PanelHeader1 控件下方，设定 ID 属性值为 PanelContent1。它是用来显示博客栏目子分类的标题，这里制作了一个无序列表，具体代码如下：

```
<asp:Panel ID="PanelContent1" runat="server">
    <ul>
    <li>
        <a href="#">
        <span class="bg">01</span><span class="nor">Top</span>硬件知识
        </a>
    </li>
    <li>
        <a href="#">
        <span class="bg">02</span><span class="nor">Top</span>路由交换
        </a>
    </li>
    <li>
        <a href="#">
        <span class="bg">03</span><span class="nor">Top</span>Windows
        </a>
    </li>
    <li>
        <a href="#">
        <span class="bg">04</span><span class="nor">Top</span>安全防护
        </a>
    </li>
    <li>
```

```
        <a href="#">
        <span class="bg">05</span><span class="nor">Top</span>LInux 社区
        </a>
    </li>
    <li>
        <a href="#">
        <span class="bg">06</span><span class="nor">Top</span>IT 认证考试
        </a>
    </li>
    </ul>
</asp:Panel>
```

下面该 CollapsiblePanelExtender 出场了。现在我们将一个 CollapsiblePanelExtender 控件放到 PanelContent1 之后，并设置相关属性值，具体代码如下：

```
<cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender1" runat="server"
  TargetControlID="PanelContent1"     <!-- 对 PanelContent1 进行收缩 -->
  ExpandControlID="PanelHeader1"        <!-- 激发伸展效果的控件 -->
  CollapseControlID="PanelHeader1"       <!-- 激发折叠效果的控件 -->
  Collapsed="False"                    <!-- 打开页面为展开状态 -->
  ImageControlID="PanelHeaderImage1"    <!-- 显示图像位置的 ID -->
  CollapsedImage="images/collapse.jpg"   <!-- 收缩时使用的图片路径 -->
  ExpandedImage="images/expand.jpg"     <!-- 展开时使用的图片路径 -->
  SuppressPostBack="true" >             <!-- 禁止回传 -->
</cc1:CollapsiblePanelExtender>
```

至此，一个标题为“IT 技术杂谈”的可收缩面板终于做完了。

下面，我们可以根据上述步骤炮制出“摄影爱好者”、“会员登录”等面板的可伸缩设计，具体内容可参考本节的源程序。需要注意的就是它们的 ID 必须唯一。

至此，博客分栏的伸缩效果我们设计完了，下面再来设计博客文章内容的折叠效果。

首先，拖一个 Accordion 控件到 form1 中 UpdatePanel1 外，设定其属性值，具体代码如下：

```
<cc1:Accordion ID="Accordion1" runat="server"
  AutoSize="None"             <!-- 自动调整大小的模式 -->
  SelectedIndex="0"           <!-- 默认显示的 AccordionPane 控件的索引号 -->
  FadeTransitions="true"          <!-- 是否在折叠展开时有渐变效果 -->
  TransitionDuration="600"        <!-- 折叠展开时过渡动画持续时间 -->
  FramesPerSecond="60"        <!-- 过渡动画每秒所需要的帧数 -->
  RequireOpenedPane="false"><!-- 单击控件 Pane 中的 header 时是否关闭打开的 Pane -->
</cc1:Accordion>
```

然后在 Accordion1 中加一对<Panes>标记，再拖 3 个 AccordionPane 控件到 Panes 集合中，并在分别在 3 个 AccordionPane 控件中添加<Header>、<Content>标记，具体内容请参考本节的源程序，AccordionPane1 的代码如下：

```
<Panes>
<cc1:AccordionPane ID="AccordionPane1" runat="server">
    <Header>
        <h3>LCD 养生之道 液晶显示器清洁保养技巧</h3>
```

```
    </Header>
    <Content>
        <p class="rightTxt1">LCD 养生之道 液晶显示器清洁保养技巧</p>
        <p class="rightTxt2">LCD 养生之道 液晶显示器清洁保养技巧</p>
    </Content>
</cc1:AccordionPane>
</Panes>
```

至此，博客主页的主体部分已设计完毕，该段代码的层次结构如图 7-23 所示。

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<div id="left">
    <asp:Panel ID="PanelHeader1">...</asp:Panel>
    <asp:Panel ID="PanelContent1">...</asp:Panel>
    <asp:Panel ID="PanelHeader2">...</asp:Panel>
    <asp:Panel ID="PanelContent2">...</asp:Panel>
    <asp:Panel ID="PanelHeader3">...</asp:Panel>
    <asp:Panel ID="PanelContent3">...</asp:Panel>
    <cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender1" ...>...</cc1:Collapsibl
    <cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender2" ...>...</cc1:Collapsibl
    <cc1:CollapsiblePanelExtender ID="CollapsiblePanelExtender3" ...>...</cc1:Collapsibl
</div>
<div id="right">
    <cc1:Accordion ID="Accordion1" runat="server" AutoSize="None" SelectedIndex="0" Fade
        TransitionDuration="600" FramesPerSecond="60" RequireOpenedPane="false">
        <Panes>
            <cc1:AccordionPane ID="AccordionPane1">...</cc1:AccordionPane>
            <cc1:AccordionPane ID="AccordionPane2">...</cc1:AccordionPane>
            <cc1:AccordionPane ID="AccordionPane3">...</cc1:AccordionPane>
        </Panes>
    </cc1:Accordion>
</div>
```

图 7-23　主体部分代码的层次结构

最后再啰嗦几句。为了让我们的页面看起来更美观一些，可以把本段代码套用到自己喜欢的博客模板。具体操作不做过多说明，读者在套用的过程中注意一下 DIV 的层次结构即可。

5. 运行结果

呵呵，忙乎了半天，很想看看战果如何，赶快保存，点击运行吧。如图 7-24 所示为将文章列表与栏目全部折叠时的效果。



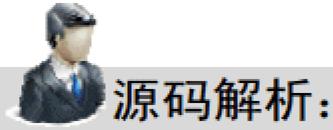
图 7-24　全部折叠时的效果

单击文章标题即可展开该文章，左侧的栏目可以同时展开多个。但同时只能查看一个文章的内容，这也是 Accordion 和 CollapsiblePanelExtender 的区别，如图 7-25 所示为部分展开时的效果。



图 7-25　部分展开时的效果

7.7.5.6　实例分析



源码解析：

从运行结果上看，Accordion 控件的效果有点类似于多个 CollapsiblePanelExtender 控件叠加起来的效果，都是控制指定区域的缩放，但 Accordion 控件管理的是一个区域群，群中每个区域之间能够产生互动，关闭一个时能打开另一个，而 CollapsiblePanelExtender 主要是控制单个区域的缩放。

另外在使用上，CollapsiblePanelExtender 控件比 Accordion 简单，能通过设置显示不同状态的图片和文字，读者在选择的时候一定要从实际出发，选择合适的组件。

7.7.6　为博客文章进行等级评分

在网站开发中，经常需要用户给某项事物评分或者进行投票。在传统的开发方式中，用户每评一次分或进行一次投票，都需要 PostBack 一次。通常情况下，评分或投票操作的数据传输是非常小的，甚至只是一个数字，整页回送的方式大大浪费了带宽和系统资源。因此，评级 Rating 控件应运而生。

在实例中将通过该控件让用户选择他们对某篇文章所要给予的等级。



视频教学：光盘/videos/7/Rating.avi　　　　**长度：8 分钟**

1. 基础知识——Rating 控件

等级选择控件 Rating 提供了一种全新的方式来进行等级选择，在以往的 Web 上我们只能

通过使用特殊字符“☆”来表达等级。有了等级选择控件 Rating，我们可以用一种更直观的等级选择控件，例如★★★★☆。只要鼠标单击“☆”图标就表示选择的等级是几级。

Rating 控件本身支持网页无刷新功能，在使用的时候完全没有必要将它放在 UpdatePanel 控件里，而只需要将它的 AutoPostBack 属性设为 True 即可。该控件使用起来非常简单，而且还可以设置各种不同的效果，以及自定义函数回调等。如下是它的语法格式：

```
<ajaxToolkit:Rating ID="Rating1" runat="server"
  CurrentRating="2" MaxRating="5"
  StarCssClass="ratingStar"
  WaitingStarCssClass="saveRatingStar"
  FilledStarCssClass="filedRatingStar"
  EmptyStarCssClass="emptyRatingStar"
  OnChanged="Rating1 Changed" >
</ajaxToolkit:Rating>
```

Rating 控件允许用户设置等级的最大值、等级控件的排列顺序(垂直/水平)、自定义“☆”表示的级别状态。还会在用户选择 Rating 后触发服务端的 OnChanged 事件。

Rating 控件常用的属性或事件如表 7-7 所示。

表 7-7　Rating 控件常用的属性或事件

属性或事件	说　明
AutoPostBack 属性	当评分时是否引用引起 PostBack，设置为 true 表示引起 PostBack
CurrentRatiing 属性	初始化默认的评价值
MaxRating 属性	最高评价值
ReadOnly 属性	用于指定是否可以评价
StarCssClass 属性	用于指定初始化未评价时的 CSS 样式
WaitingStarCssClass 属性	用于指定鼠标停留在控件上，准备控件评价的 CSS 效果
FilledStarCssClass 属性	用于指定已评价选中的效果，如评为 2 星，那么这 2 星的效果由此指定
EmptyStarCssClass 属性	已平价未选中的效果，如评为 2 星，那么其余 3 星的效果由此指定
RatingAlign 属性	用于评价星星的排列顺序，可选水平或者垂直
RatingDirection 属性	用于评价星星的排列方面，可以是从左到右从上到下或从右到左从下到上
ScroolBar 属性	用于指定是否出现滚动条，若有出现什么样的滚动
Tag 属性	用于指定给 ClientCallBack 事件传递的参数
OnChanged 事件	当做出评价时触发该事件

2. 实例描述

在 Web 页面中，常见的让用户评定等级的方法是使用一系列的代表不同评分的单选按钮，如图 7-26 所示为某网站产品的评分系统。

用户需要选择几个单选项来表示不同的评级，然后提交表单。固然，对于开发人员而言，这种做法无可厚非，毕竟他实现了功能需求；对于用户来讲，因为熟悉了这种界面，也不会有什么怨言。

不过在看到了 Windows Media Player 中音乐库的评级功能之后，你一定会立刻喜欢上这种直观的实现方式！图 7-27 显示了 Windows Media Player 中音乐列表的界面。



图 7-26　某网站产品的评分系统

	标题	长度	分级	参与创作...
1	01. 忘情水	4:25		刘德华
2	02. 缠绵	4:35		刘德华
3	03. 不该爱上你	4:12		刘德华
4	04. 错怪	5:01		刘德华
5	05. 心酸	4:07		刘德华
6	06. 你是我的温柔	4:27		刘德华
7	07. 痴	4:23		刘德华
8	08. 最孤单的人是我	4:34		刘德华
9	09. 峰回路转	4:29		刘德华
10	10. 想要飞	4:07		刘德华

图 7-27　Windows Media Player 中的音乐列表

可以看到，图 7-27 中的每一首音乐都包含了一个最高 5 颗星级的评价选项。当前的评定级别用实心的星形图标表示，其他星形图标则为空心样式。当用户的鼠标悬浮在用来评级的 5 颗星之上时，会自动突出显示当前以及所有先前的星形图标，给用户以充分的提示。若用户希望对某首音乐评级，只要在相应的星形图标上点击鼠标即可，非常直观明了。

下面，让我们继续以 Blog 网站为平台，对博客文章添加一个等级评分功能，体验一下 Rating 控件与传统网页中评分效果的不同，轻松实现类似图 7-27 所示的效果。

3. 实例应用

【例 7-12】 为博客文章进行等级评分。

打开前面制作的博客首页，向 AccordionPane1 中添加一个 UpdatePanel 控件，位置为 Content 模板中。完成的代码如下：

```
<cc1:AccordionPane ID="AccordionPane1" runat="server">
    <Header><h3>LCD 养生之道 液晶显示器清洁保养技巧</h3></Header>
    <Content>文章内容省略……
        <div>
            <asp:UpdatePanel ID="UpdatePanel2" runat="server">
            </asp:UpdatePanel>
        </div>
    </Content>
</cc1:AccordionPane>
```

接下来，我们向 UpdatePanel2 中拖放两个 Label 控件，并设定相关的属性值，代码如下：

```
<asp:UpdatePanel ID="UpdatePanel2" runat="server">
    <ContentTemplate>
        <asp:Label ID="Label1" runat="server" Text="请评分！"></asp:Label>
        <asp:Label ID="Label2" runat="server" ForeColor="Red"></asp:Label>
        <!-- 用于显示评分结果 -->
    </ContentTemplate>
</asp:UpdatePanel>
<br/><hr style="border:1px solid #000000;" />
```

紧接着 Label2，我们拖放一个 Rating 控件，并设定相关属性值，代码如下：

```
<cc1:Rating ID="Rating1" runat="server"
  CurrentRating ="2"                  <!-- 初始化时的默认评价值 -->
  MaxRating ="6"                      <!-- 最高评价值 -->
  StarCssClass="RatingStar"               <!-- 初始化未评价时的 CSS 样式 -->

  WaitingStarCssClass="savedRatingStar"  <!-- 鼠标停留在控件上，
                                            准备控件评价的 CSS 效果 ->

  FilledStarCssClass="filledRatingStar"   <!-- 已评价选中星的效果 -->
  EmptyStarCssClass="emptyRatingStar"  <!-- 已评价未选中星的效果 -->
  onchanged="Rating1_Changed"         <!-- 当做出评价时触发该事件 -->
  AutoPostBack ="true" >              <!-- 当评分时是否引用引起 PostBack -->
</cc1:Rating>
```

在上面的代码中，我们指定了 Rating 控件在各种状态下的 CSS 样式。下面就来编写这些 CSS 代码，具体代码如下：

```
<style type="text/css">
    .RatingStar{
            width:20px; height:15px;
            margin:0px; padding:0px;
            cursor:pointer; display:block;
            background-repeat:no-repeat;
        }
    .filledRatingStar {
            background-image:url(images/FilledStar.png);
        }
    .emptyRatingStar {
            background-image:url(images/EmptyStar.png);
        }
    .savedRatingStar {
            background-image:url(images/SavedStar.png);
        }
</style>
```

在 Rating 控件中，我们指定了它的 onchanged 的属性值为 Rating1_Changed。这一步，我们来编写 Rating1_Changed 事件的代码，具体代码如下：

```
protected void Rating1_Changed(object sender,
  AjaxControlToolkit.RatingEventArgs e)
{
   this.Label2.Text = "您为该篇文章评了" + Rating1.CurrentRating.ToString() + "星! ";
}
```



在使用 rating 控件时，我们发现点击 rating 控件时页面会产生跳动！只要在 page_load 里加上下面一句代码就可以避免了：

```
Rating1.Attributes.Add("onclick", "return false;");
```

4. 运行结果

至此 Rating 控件打造的文章等级评论实例就完成了，我们来运行页面查看一下效果。如图 7-28 所示是默认未进行评分时的效果。

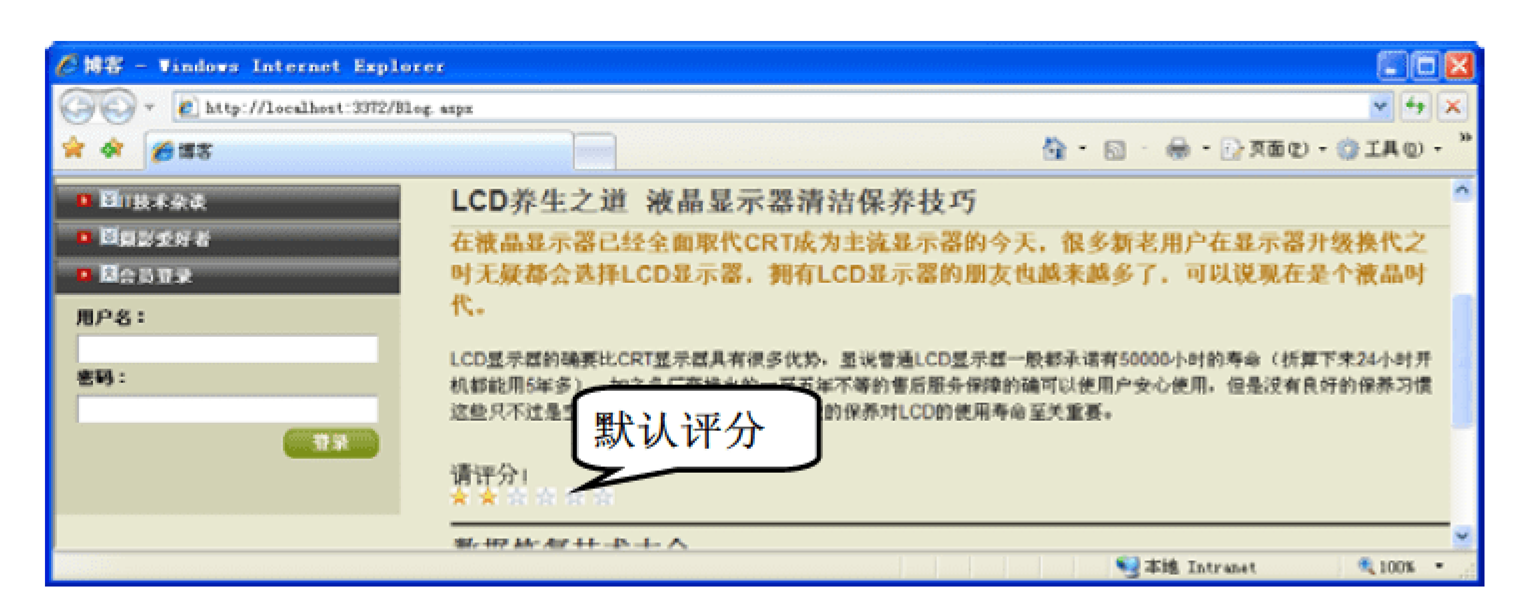


图 7-28　未评分时的效果

从图 7-28 中我们看到，初始状态下亮了两颗星，表明我们设定的默认评分等级为 2。然后将光标移动到其他星上，星的颜色发生了改变，当单击星的时候，会提示我们为该篇文章评分的相应等级。这时候可以发现页面并没有刷新，效果如图 7-29 所示。



图 7-29　评分之后的效果

5. 实例分析



源码解析：

整体来看，本实例并没有多大难度，读者在实际应用中应注意以下两点。

(1) AutoPostBack 属性的设置。若该 Rating 控件只提供评价的功能，则应该将其设置为 false，这样可以充分利用 Rating 自带的异步回调功能。但若做出评价后还需要改变其他控件的一些特征，则必须将 AutoPostBack 属性设置为 true，并用 UpdatePanel 包围该控件，以得到 Ajax 的异步回送功能。

(2) 实际应用时将 Rating 控件的 CurrentRating 值与相应数据库的表字段值进行绑定即可。

7.7.7　博客后台分类管理

CascadingDropDown 控件可以控制多个 DropDownList 控件，并使它们产生无刷新的级联效果。默认的填充数据源是基于 XML 的，它是通过 Web Service 去调用 XML 文件，通过节点来区分子节点，然后加载相关的数据。

实例以加载 XML 数据源的方式来学习此控件的用法，实现添加博客文章时选择类别的二级级联下拉框。



视频教学：光盘/videos/7/Casecad.avi　　　　　　　　**长度：**13 分钟

1. 基础知识——CasecadingDropDown 控件

CascadingDropDown 控件用于级联下拉列表的选择，也就是当未选择第一级下拉列表时，第二级下拉列表不可用，当选择了第二级列表时，第三级才可用，依次类推。这种效果通常实现在需要多个选择的时候，例如只有选择了省份后，才可以选择省份下的城市。

CascadingDropDown 控件的使用格式如下所示：

```
<ajaxToolkit:CascadingDropDown ID="CDD1" runat="server"
  TargetControlID="DropDownList2"
  Category="Model"
  PromptText="Please select a model"
  LoadingText="[Loading models...]"
  ServicePath="CarsService.asmx"
  ServiceMethod="GetDropDownContents"
  ParentControlID="DropDownList1"
  SelectedValue="SomeValue">
</ajaxToolkit:CascadingDropDown>
```

在上述语法格式中，常用属性的说明如表 7-8 所示。

表 7-8　CasecadingDropDown 控件的常用属性

属　性	说　明
TargetControlID	对应的下拉列表控件的 ID
Category	对应的数据的分类，例如“区域”、“省份”或者“城市”
PromptText	下拉列表中无数据或未选择数据时给用户的提示
LoadingText	加载下拉列表数据时的提示
ServicePath	获取数据的 Web Service 的路径，为每个下拉列表都要指定
ServiceMethod	下拉列表获取数据所需要的 Web 服务方法
ParentControlID	控制此下拉列表控件的父级控件
SelectedValue	默认选择项的值，这里指的是 DropDownList 中 ListItem 的 Value 值

CasecadingDropDown 控件的 ServiceMethod 属性非常重要，它对应一个 Web Service 方法名，而且必须指定为如下的方法签名形式：

```
[System.Web.Services.WebMethod]
[System.Web.Script.Services.ScriptMethod]                    //方法签名
public CascadingDropDownNameValue[] GetDropDownContents(    //方法声明
  string knownCategoryValues,                               //方法参数
  string category)
{  //具体方法的实现代码 }                                      //方法实体
```

2. 实例描述

前些天做了一个网站后台，在添加文章这一块，对文章的分类一直找不到好的处理方法。我们的博客内容暂时分为两大类，每一大类里面又有若干小类，若直接采用两个下拉菜单现实的话，也没有什么不妥。但是，若以后还想再添加新的栏目，栏目里面想添加新的分类的话，就会发现下拉菜单会越来越长，而且后台也变得越来越不易维护。

难道就没有好的解决办法了吗？答案是肯定有的。通过上面我们对 CasecadingDropDown 控件的学习，发现一句话是多么的经典——“踏破铁鞋无觅处，得来全不费工夫”，这不正是我们要找的吗。好了，不多说了，赶快拿来用吧！

3. 实例应用

【例 7-13】 博客后台分类管理。

我们的网站后台布局如图 7-30 所示，现在要做的就是把它的 DropDownList 和 Cascading DropDown 控件关联起来，实现级联效果。



图 7-30　后台布局

赶快在 Visual Studio 2008 中打开我们的网站后台开始体验吧。要用到 CascadingDropDown 控件，所以还得先请出两位老大哥——ScriptManager 控件和 UpdatePanel 控件，代码如下：

```
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <table>布局参考本节源码……</table>
            <table>布局参考本节源码……</table>
            <table>布局参考本节源码……</table>
        </ContentTemplate>
    </asp:UpdatePanel>
</form>
```

进入“源码”视图，找到 DropDownList，我们紧随其后拖入两个 CascadingDropDown 控件，并设定相关的属性值。

完成后的具体代码如下：

```
<td valign="top" align="left"
  style="background-color: #C4D8ED; color: #135294; padding: 5px;">
    <asp:DropDownList ID="DropDownList1" runat="server" Width="150px">
    </asp:DropDownList>
    <asp:DropDownList ID="DropDownList2" runat="server" AutoPostBack="true"
    OnSelectedIndexChanged="DDL2 SelectedChanged" Width="150px">
    </asp:DropDownList>
    <cc1:CascadingDropDown ID="CascadingDropDown1" runat="server"
      Category="project"                     <!-- 对应数据的分类 -->
      PromptText="请选择文章分类"              <!-- 选择前用户提示信息 -->
      ServicePath="CascadingDropDownDemo.asmx" <!--获得数据的 Web Service 路径-->
      ServiceMethod="GetDropDownContents"      <!-- 获得 Web Service 方法名 -->
      TargetControlID="DropDownList1">        <!-- 扩展的 DropDownList 控件 ID -->
    </cc1:CascadingDropDown>
    <cc1:CascadingDropDown ID="CascadingDropDown2" runat="server"
      Category="detail"
      ParentControlID="DropDownList1"     <!-- 上一级 DropDownList 控件的 ID -->
      PromptText="请选择文章类型"
      ServiceMethod="GetDropDownContents"
      ServicePath="CascadingDropDownDemo.asmx"
      TargetControlID="DropDownList2">
    </cc1:CascadingDropDown>
</td>
```



属性 TargetControlID、ParentControlID、ServicePath 和 ServiceMethod 是必选的，其他各属性用户可以根据需要进行选择。需要注意的是当其没有上一级 DropDownList 的时候，ParentControlID 属性可以不设。

在上述代码中，使用了两个 CascadingDropDown 控件分别与两个 DropDownList 控件进行绑定，并通过属性设置它的目标控件和父控件，同时指定了所用到的 Web 服务及其相关方法。这里 Web 服务的实现文件为 CascadingDropDownDemo.asmx。

它的源码文件(CascadingDropDownDemo.cs)部分代码如下：

```
using System.Xml;
using System.Collections.Specialized;
...
public class CascadingDropDownDemo : System.Web.Services.WebService
{
    private static XmlDocument  document;
    private static object _lock = new object();
    public static XmlDocument Document
    {
        get
        {
```

```
            lock (_lock)
            {
                if ( document == null)
                {
                    document = new XmlDocument();
                    document.Load(HttpContext.Current.Server
                     .MapPath("~/App Data/CascadingDropDownDemo.xml"));
                }
            }
            return  document;
        }
    }
    public static string[] Hierarchy
    {
        get { return new string[] { "project"}; }
    }
    [WebMethod]
    public AjaxControlToolkit.CascadingDropDownNameValue[]
     GetDropDownContents(string knownCategoryValues, string category)
    {
        StringDictionary  knownCategoryValuesDictionary =
          AjaxControlToolkit.CascadingDropDown
          .ParseKnownCategoryValuesString(knownCategoryValues);
        return AjaxControlToolkit.CascadingDropDown
          .QuerySimpleCascadingDropDownDocument(Document, Hierarchy,
          knownCategoryValuesDictionary, category);
    }
}
```

如果我们想要在页面中显示选择的结果，那么只需在网页的适当位置拖放一个 Label，另外给最后一个下拉列表添加一个 OnSelectedIndexChanged 事件即可，它触发后台处理文件 CascadingDropDownDemo.aspx.cs 中的方法 DDL2_SelectedChanged ()。事件代码如下：

```
protected void DDL2 SelectedChanged(object sender, EventArgs e)
{
    string project = DropDownList1.SelectedItem.Text;
    string detail = DropDownList2.SelectedItem.Text;
    if (string.IsNullOrEmpty(project))
    {
        Label2.Text = "请选择文章分类";
    }
    else if (string.IsNullOrEmpty(detail))
    {
        Label2.Text = "请选择文章类型";
    }
    else
    {
        Label2.Text = string.Format("\"{0}>>{1}\"栏目", project, detail);
    }
}
```

关于级联菜单中数据的来源，我们可以使用 XML 文档，也可以使用数据库，这里我们使用 XML 文档。

如下代码为本实例中使用的 CascadingDropDownDemo.xml 文件内容：

```
<?xml version="1.0" encoding="utf-8" ?>
<case>
    <project value="itjs" name="IT 技术" >
        <detail name="交换\路由"/>
        <detail name="硬盘\主板"/>
        <detail name="Windows 系统优化"/>
        <detail name="软件破解"/>
        <detail name="Linux 论坛"/>
    </project>
    <project value="smsy" name="数码摄影" >
        <detail name="单反镜头"/>
        <detail name="三角架"/>
        <detail name="新手上路"/>
        <detail name="摄影技巧"/>
    </project>
</case>
```



需要将页面的 EnableEventValidation 属性值设置为 false，表示在页面执行时关闭事件数据验证。

4. 运行结果

经过以上几步改造，我们来看看 CascadingDropDown 控件给我的下拉列表带来了怎样的全新面貌。

保存一下文件，在浏览器中的效果如图 7-31 所示。



图 7-31　运行效果

在图 7-31 中，我们看到，第二个下拉列表不可用，当我们在第一下拉列表中选择一项时，它改变颜色，变成可用，并且该下拉列表中列出与第一下拉列表选中项相关的选项，如图 7-32 所示。也就是说，CascadingDropDown 控件实现了我们预期的效果。



图 7-32 操作后的效果

5. 实例分析



源码解析：

本实例中，CascadingDropDown 控件的使用并没什么难度，很容易掌握。但从 XML 文件中获取数据的方法需要好好理解一下。在 Web 服务的实现文件 CascadingDropDownDemo.asmx 代码中，静态方法 Document 用于返回 XmlDocument 对象；静态方法 Hierarchy 用于返回 XML 文件的层次；而 GetDropDownContents()方法用于返回列表数据。

这里需要注意的是，GetDropDownContents()方法的参数 knownCategoryValues 和 category 不能被改变名称。因为 AjaxControlToolkit 所规定的签名格式，包括参数字母大小写、数据类型都是不可变的，如果要更改必须重新修改 AjaxControlToolkit 源码。

7.7.8 仿 Baidu 的自动完成功能

本实例将通过使用 Ajax 框架提供的 AutoCompleteExtender 控件来实现仿 Baidu 自动完成功能。AutoCompleteExtender 控件通过 Web Service 传递数据，在传递参数的过程中对格式有一定要求。

下面就来看看具体的实现过程与方法。

视频教学： 光盘/videos/7/AutoComplete.avi　　**长度：** 12 分钟

1. 基础知识——AutoCompleteExtender 控件

AutoCompleteExtender 控件可以辅助 TextBox 控件自动地完成输入，它必须与 Web Service 相连接才能发挥作用。运行原理是：当用户输入一些字符后，AutoCompleteExtender 自动异步调用相应的 Web 服务，并取得相关的数据，然后以下拉框的形式显示在输入框的下面供用户进行选择。

AutoCompleteExtender 控件的使用格式如下：

```
<ajaxToolkit:AutoCompleteExtender runat="server" ID="AutoComplete1"
  TargetControlID="myTextBox"
  ServiceMethod="GetCompletionList"
  ServicePath="AutoComplete.asmx"
  MinimumPrefixLength="2"
  CompletionInterval="1000"
  EnableCaching="true"
  CompletionSetCount="20"
  CompletionListCssClass="autocomplete completionListElement"
  CompletionListItemCssClass="autocomplete listItem"
  CompletionListHighlightedItemCssClass="autocomplete highlightedListItem"
  DelimiterCharacters=";, :">
    <Animations>
        <OnShow> ... </OnShow>
        <OnHide> ... </OnHide>
    </Animations>
</ajaxToolkit:AutoCompleteExtender>
```

在上述格式中，属性 TargetControlID、ServicePath 和 ServiceMethod 是必选的，其他各属性用户可以根据需要进行选择。

表 7-9 给出了该控件的常用属性或事件。

表 7-9　AutoCompleteExtender 控件所常用的属性或事件

属性或事件	说　明
TargetControlID 属性	指定将被辅助完成自动输入的控件 ID，这里的控件只能是 TextBox
ServiceMethod 属性	指定在 Web Service 中的用于提取数据的方法的名称
ServicePath 属性	提供 Web Service 的路径，默认表示 ServiceMethod 本页面对应的方法名
MinimumPrefixLength 属性	用来设置用户输入多少字符才会出现相应的提示信息，默认值为 3
CompletionInterval 属性	从服务器获取数据的时间间隔，单位是毫秒，默认值为 1000ms
EnableCaching 属性	用于指定是否提供客户端缓存数据的功能，默认为 true
CompletionSetCount 属性	用于指定自动完成显示的条数，默认值为 0
CompletionListItemCssClass 属性	自动完成的下拉列表项的 CSS 样式
CompletionListCssClass 属性	自动完成的下拉列表的 CSS 样式
<Animations>事件	指定自动完成下拉列表的显示或隐藏时的动画效果

仅使用表 7-9 中的属性还不足以实现功能，该控件对调用的 Web Service 也有一定要求。首先由于该 Web Service 是为 Ajax 框架提供服务的，因此在类声明之前需要添加属性声明 [System.Web.Script.Services.ScriptService]。

另外需要注意的是，为 AutoCompleteExtender 控件提供服务的方法(即 ServiceMethod 属性指定的值)都必须完全满足以下 3 个条件：

- 方法的返回类型必须为 string []。
- 方法的传入参数类型必须为 string 和 int。
- 两个传入参数名必须为 prefixText 和 count。



AutoCompleteExtender 可以使用客户端缓存，使得同样的请求只需要用一次 Web 服务，从而提高了反应速度。

2. 实例描述

用过 Baidu 的读者都会发现，当我们在搜索框输入关键字的时候，Baidu 页面会自动列出相关关键字提示。那么用 Ajax 框架也能实现这个人性化的功能么？

答案是：当然啦！

AutoCompleteExtender 控件的使命就是协助用户实现这个功能，不但能为用户提供更好的体验，同时也为网站添彩不少。

还是让我们通过实例来体验 AutoCompleteExtender 控件的神奇魅力吧！

3. 实例应用

【例 7-14】仿 Baidu 的自动完成。

(1) 在 VS2008 中新建一个 KaiXin.aspx 页面。

(2) 关于布局，这里不再做过多的介绍，可以参考 Baidu 的首页进行布局，如图 7-33 所示为实例中的最终效果。下面将对本实例中用到的控件进行详细介绍。



图 7-33　Autofinish.aspx 页面的布局效果

(3) 从控件工具箱中依次添加一个 ScriptManager 控件和一个 UpdatePanel 控件到 form 表单里面，代码如下：

```
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
        <ContentTemplate>
            <!-- 在此处添加布局 -->
        </ContentTemplate>
    </asp:UpdatePanel>
</form>
```

(4) 然后添加一个 TextBox 文本框和一个 Input 提交按钮到 UpdatePanel 控件中，并设置相关属性值，代码如下：

```
<ContentTemplate>
<asp:TextBox ID="myTextBox" runat="server" maxlength="100" Width="390px"/>
```

```
<input type="submit" value="越搜越开心" ID="su"/>
</ContentTemplate>
```

(5) 接下来将 AutoCompleteExtender 控件放到 UpdatePanel 控件中，设置其 TargetControlID 属性为 myTextBox，最终代码如下：

```
<cc1:AutoCompleteExtender ID="AutoCompleteExtender1" runat="server"
  TargetControlID="myTextBox"
  CompletionSetCount="10"
  MinimumPrefixLength="1"
  ServiceMethod=" GetCompleteList "
  ServicePath="WebService.asmx">
</cc1:AutoCompleteExtender>
```

如上述代码所示，在 AutoCompleteExtender 控件中还指定列出输入提示的数量、Web 服务的文件名及调用 Web 服务的函数名。



属性 TargetControlID、ServicePath 和 ServiceMethod 是必选的，对于其他各属性，用户可以根据需要进行选择。

(6) 在以上几个控件添加完成后，我们整个页面的布局和设计也就完成了。此时页面效果如图 7-34 所示。

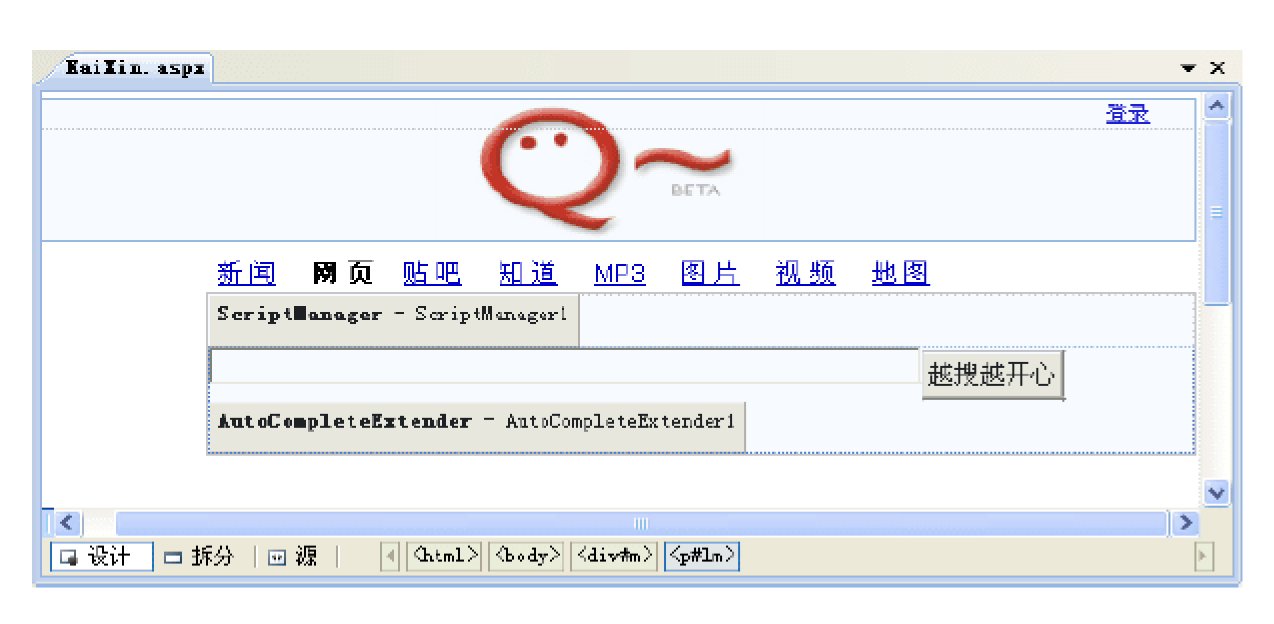


图 7-34　Autofinish.aspx 页面的最终效果

(7) AutoCompleteExtender 控件需要搭配 Web Service 才能工作。为此，我们要先创建一个“Web 服务”文件。

(8) 将文件命名为“WebService.asmx”，创建 GetCompleteList()方法并编写实现自动完成功能的代码，具体代码如下：

```
[WebMethod]
public string[] GetCompleteList(string prefixText, int count)
{
    string strSQL = "select serviceId,serviceName From [service]
      where  serviceName like '%" + prefixText + "%'";  //查询语句
    string connectionString =
      ConfigurationManager.ConnectionStrings["ConnStr"].ConnectionString;
    SqlConnection conn = new SqlConnection(connectionString);   //建立连接
    conn.Open();                                               //打开连接
    DataSet ds = new DataSet();
```

```
    SqlDataAdapter da = new SqlDataAdapter(strSQL, conn);
    da.Fill(ds, "ds");                              //获得结果集
    DataTable dt = ds.Tables[0];
    int count2 = dt.Rows .Count ;
    if(dt.Rows.Count > count)                       //判断是否限定返回数量
    {
        count2 = count;
    }
    string []result = new string[count2];
    for (int i=0; i<count2; i++)
    {
        result.SetValue(dt.Rows[i][1], i);
    }
    return result;                                  //返回结果
}
```

此处 GetCompleteList()方法必须与 AutoCompleteExtender 控件中 ServicePath 属性值保持一致。另外，声明[System.Web.Script.Services.ScriptService]也一定不能少，全靠它告诉 ASP.NET 允许这个 Web Service 从客户端调用。

4．运行结果

至此，各个文件终于创建完毕了。现在我们是不是有些迫不及待地想体验一下 AutoCompleteExtender 控件的神奇魅力呢？那就赶快保存文件，点击运行吧！

嗯，页面布局还真的与 Baidu 很像，但效果呢？还是输一个词来验证一下吧，在这里我们输入“开心”，效果如图 7-35 所示。哈哈，效果还真的和 Baidu 不相上下呀！

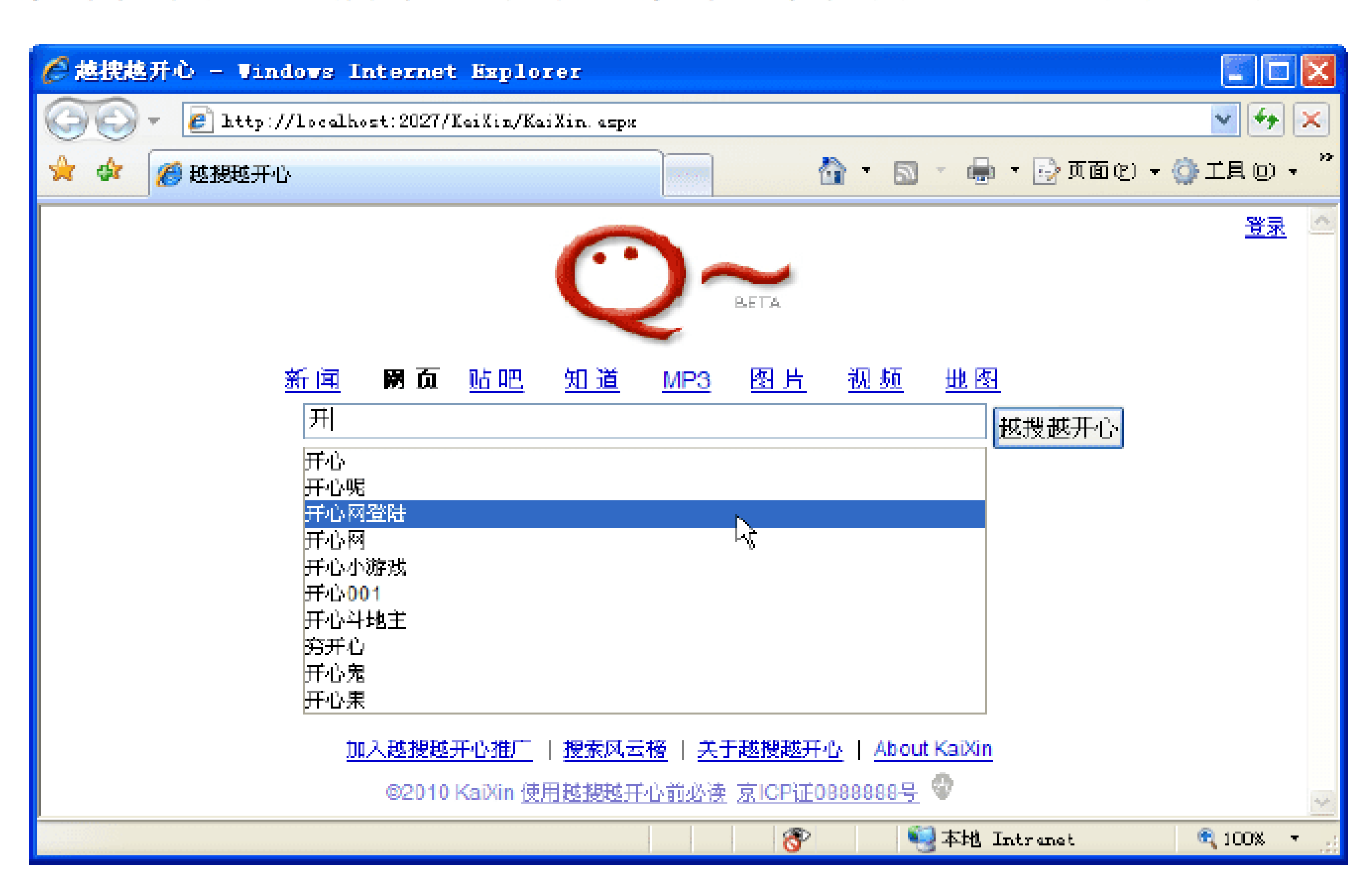


图 7-35　输入“开”字后页面的效果

5．实例分析



源码解析：

通过这个实例，我们体验了 AutoCompleteExtender 控件的用法和效果。其实，要想掌握

AutoCompleteExtender 控件并不难，在使用的过程中多注意一下其关键属性值的设定即可。

我们要重点掌握 Web Service 的编写及返回的处理。特别是 GetCompleteList()方法的传入参数类型、传入参数名以及返回值类型。

7.8　常见问题解答

7.8.1　Ajax 中的 Get 与 Post 的问题



Ajax 中的 Get 与 Post 的问题。
网络课堂：http://bbs.itzcn.com/thread-2843-1-1.html

我用 ASP.NET 做服务器端，Ajax 中 Get 与 Post 好像是一样的，调用 send()方法的时候 url 都没有附加参数，我试过了，这是为什么呀？

此外，无论用什么方法，浏览器的地址栏都没有变化，这是为什么呢？

【解决办法】：Post 和 Get 这两种提交方式您清楚吗？Get 方式传参是这样的：

```
index.aspx?id=1&name=somboy
```

在 C#中可以这样取值：

```
Request.QueryString["id"]
Request.QueryString["name"]
```

Post 方式传参在 Ajax 中这样用：

```
xmlhttp.send("id=1&name=somboy");
```

取值时用：

```
Request.Form["id"]
Request.Form["name"]
```

这下看懂了吧。如果不想严格用这两种方式传参，还可以这样获取：

```
Request["id"]
Request["name"]
```

Ajax 是异步调用的。不是同步的。如果你用一个 Form 表单提交的话，那地址栏里面的地址就变了啊。

7.8.2　UpdateProgress 何时执行的问题



UpdateProgress 何时执行的问题。
网络课堂：http://bbs.itzcn.com/thread-2859-1-1.html

有关 UpdateProgress 何时执行的问题。请问什么事件在触发 UpdateProgress 的显示？

【解决办法】：没有具体的哪一个事件一定会触发 UpdateProgress 执行的。事实上你把它放到 UpdatePanel 里它自然就会被触发了。例如，它和一个 Repeater 一起在一个 UpdatePanel 里，外面做查询显示数据的时候 UpdateProgress 会被执行。

7.9 习　　题

一、填空题

(1) Ajax 的英文全称是 Asynchronous JavaScript And XML，那么实现时它的核心是__________。

(2) XMLHttpRequest 对象的 ReadyState 属性值为__________时表示“未初始化”状态，此时已经创建一个 XMLHttpRequest 对象，但是还没有初始化。属性值为__________时表示“已加载”状态，此时响应已经被完全接收。

(3) 使用 JavaScript 在 IE 浏览器中操作 XML 时，必须先创建一个 XML 对象，代码为__________。该对象有一个__________方法可以异步载入 XML 文档。

(4) 假设有一个字符串 str，现在需要将它异步提交到 Server.aspx，请完善下面的代码:

```
createXMLHttpRequest();
var url = __________;
XmlHttp. __________ = handleStateChange;
XmlHttp.open("POST", url, true);
XmlHttp.send(__________);
```

(5) 在一个 OA 系统中要使用 Ajax 功能，需要先引用__________控件，然后在模块中可以用__________控件引用 Ajax 功能。

二、选择题

(1) 下面的哪行语句执行后不能创建一个 XMLHttpRequest 对象？__________

A. var XmlHttp = new ActiveXObject(“Msxml2.XMLHTTP”);

B. var XmlHttp = new ActiveXObject(“Microsoft.XMLHTTP”);

C. var XmlHttp = new ActiveXObject(“Windows.XMLHTTP”);

D. var XmlHttp = new XMLHttpRequest();

(2) 根据下面的描述，从选项中选择正确的正则表达式。

① 验证邮政编码。__________

② 验证用户名，要求必须是 6 至 12 个字母或者数字的组合。__________

③ 验证电话号码。__________

④ 验证密码，要求第 1 位必须是数字，后面跟字母或者数字，长度在 5 至 12 之间。__________

A. /^((\d{3,4}-)|(\d{3,4}))\d{5,9}$/　　B. /^(\w){6,12}$/

C. /^[0-9]{1}([A-Za-z0-9]|[._]){5,12}$/　　D. /^[1-9]\\d{5}(?!\\d)/

(3) 在 Ajax 的运行周期中，我们可以用__________属性来监听服务器端的回调事件。

A. onreadystatechange　　B. status

C. readyState　　D. statusText

(4) status 属性的___________值表示请求已成功并有返回值。

A. 200　　B. 202　　C. 101　　D. 400

(5) 下面关于 Accordion 控件的描述，错误的是___________。

A. 可以实现同时展开多个面板

B. 只以实现同时展开一个面板

C. 每一个 Accordion 控件可以包括若干个 AccordionPane 控件

D. 所有的子面板必须放在 Panes 集合中

(6) 下面不属于 AutoCompeteExtender 控件对 Web Service 要求的是___________。

A. 方法的返回类型必须为 string[]

B. 方法的传入参数类型必须为 string 和 int

C. 两个传入参数名必须为 prefixText 和 count

D. 方法的名称必须为 GetAllWords()

三、上机练习

上机练习 1：以 GET 和 POST 方式发送数据。

本次练习要求读者使用 XMLHttpRequest 对象获取 HTML 表单中的姓名、邮箱和网址 3 项，然后分别以 Get 和 Post 方式发送到服务器端 ASPX 页面，最终的运行效果如图 7-36 所示。

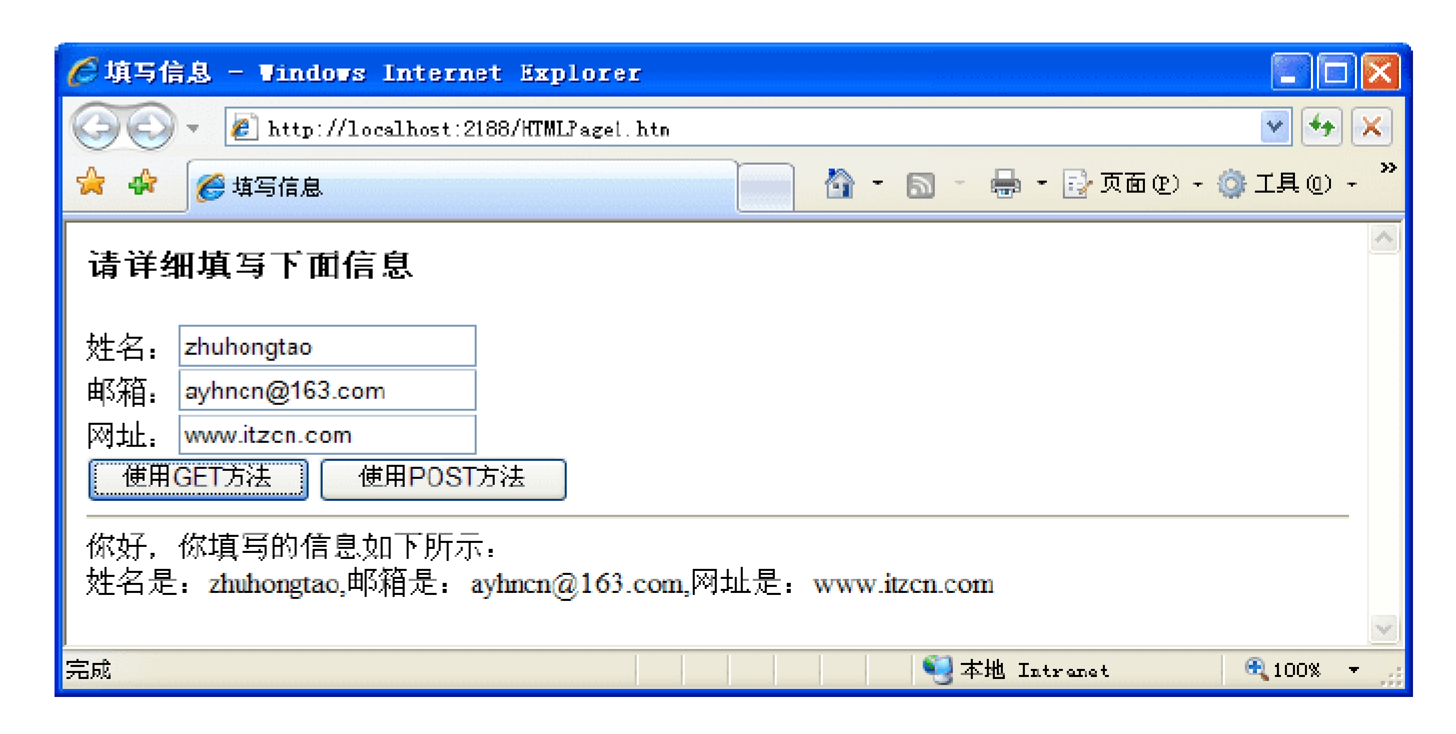


图 7-36　运行效果

上机练习 2：处理 Ajax 加载动画。

由于 Ajax 对页面是无刷新的，因此用户根本感觉不到整个交互过程。而在数据量比较大时，就需要做一个等待信息来提示用户。

UpdateProgress 控件正是为了解决此问题而设计的。

这一次练习的目的是掌握该控件的使用方法，将它与一个按钮进行组合。单击后处于加载时效果如图 7-37 所示，完成后效果如图 7-38 所示。

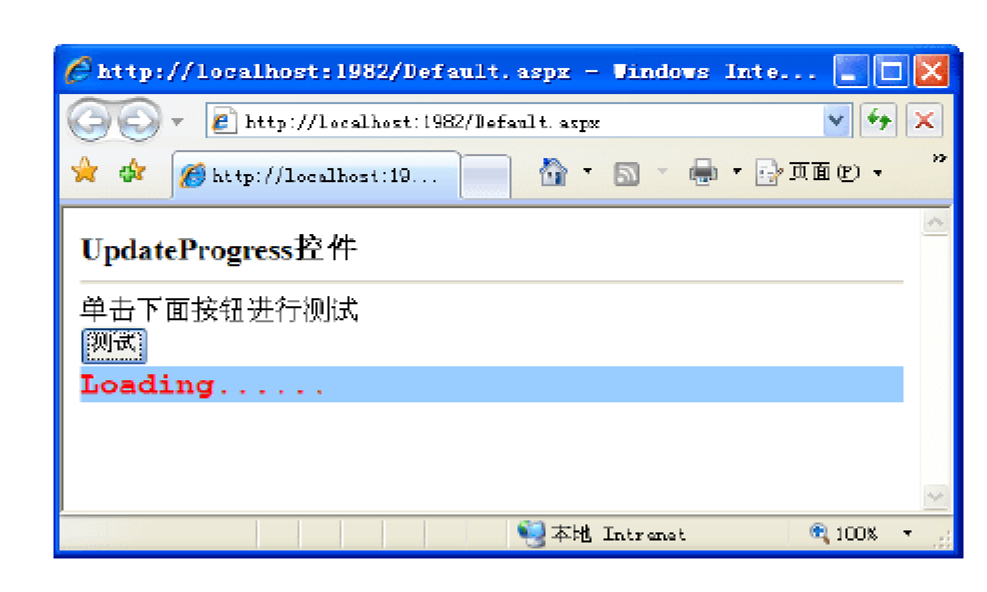


图 7-37　处理加载时的效果

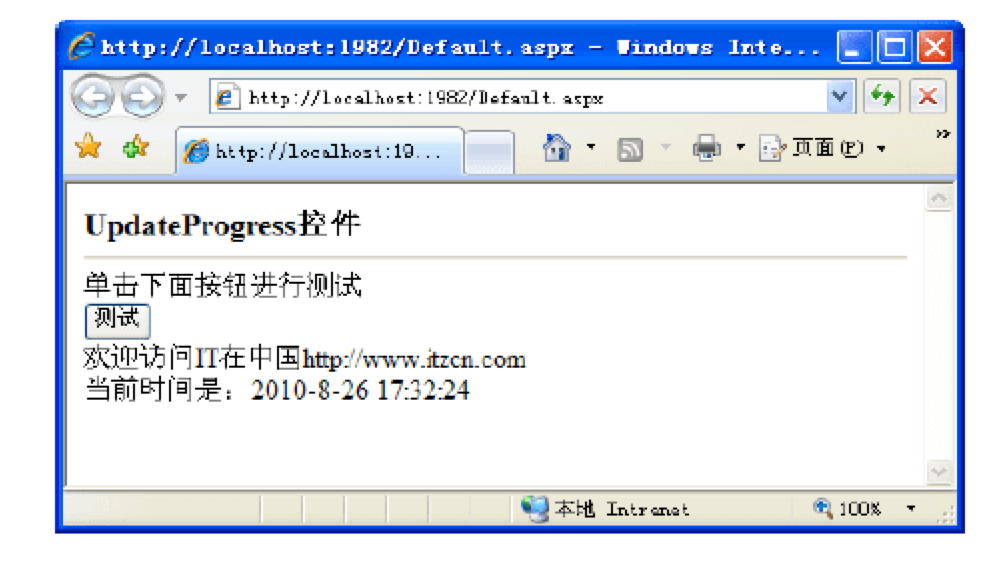


图 7-38　加载完成后的最终效果

上机练习 3：单词自动匹配。

本次练习要求读者使用 Ajax 框架提供的 AutoCompleteExtender 控件实现单词自动匹配功能。最终运行效果如图 7-39 所示，要注意该控件对 Web Service 的要求，以及在传递参数的过程中 prefixText 和 count 这两个参数名称要严格遵守拼写格式。

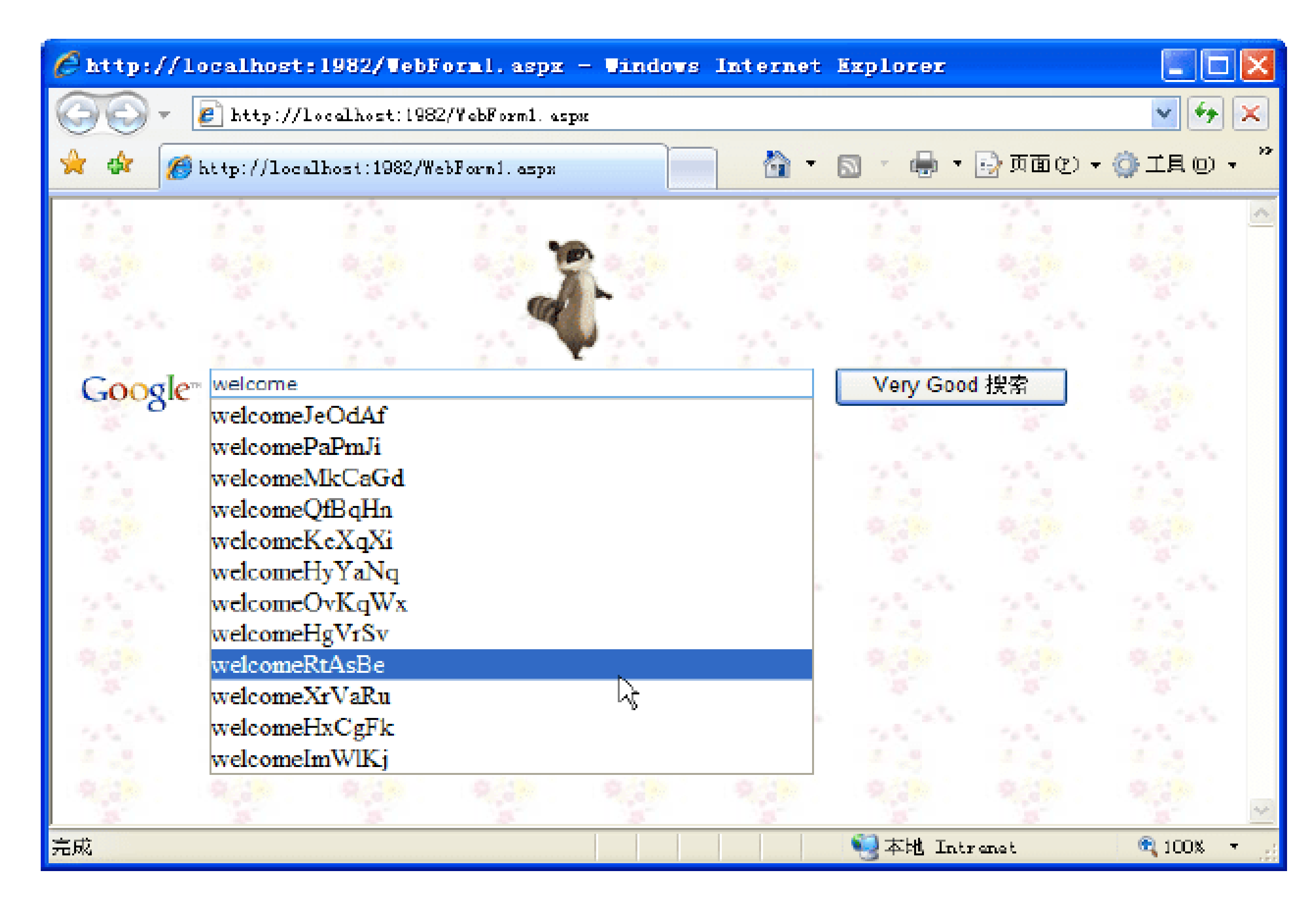


图 7-39　单词自动匹配的运行效果



第 8 章　我的MVC框架我精通

内容摘要：

MVC 首先是一种设计模式。早在 1979 年已经被提出并深入研究。随着近些年软件项目越来越庞大，明确和分离软件各部分的职责变得越来越重要。MVC 模式也被人们越来越多地注意到并大规模地应用到实际软件项目中。

作为后起之秀的 ASP.NET，虽然其特色的 WebForm 已经做得相当优秀，但它高度的封装性和制作高性能网站的效率瓶颈导致微软公司不得不考虑引入另一种先进的设计思想——MVC，来解决 WebForm 中经常出现的这样或那样的问题，这就是本章我们要学习的内容。

学习目标：

- 了解 MVC 设计模式思想
- 掌握 ASP.NET MVC 框架的特点和程序结构
- 熟练使用 URLRouting 配置站点路由
- 熟练使用数据呈现字典 ViewData 和 TempData
- 熟练使用各种提交数据的方法
- 掌握过滤器(Filter)的使用方法
- 熟练使用控件辅助工具 Helper
- 掌握 ASP.NET MVC 下的 Ajax 应用

8.1 我的第一个MVC项目

作为专业的软件开发人员，搭建开发环境是最基本的能力。理论上这一节的内容非常简单，环境搭建对于任何一个稍有些经验的开发人员都不是问题。不过借环境搭建和环境测试这个空闲，我们可以来学习一些 MVC 模式和 ASP.NET MVC 框架的基础知识，比如设计思想、运行原理之类的。

 **视频教学：光盘/videos/8/FirstMVC.avi** **长度：10 分钟**

8.1.1 基础知识——MVC模式和MVC框架

1. MVC模式

学习之前，我们得先弄明白什么是 MVC。

脱离任何一种开发语言来讲，MVC 是一个纯粹的设计模式、一种思想，也算是一种程序模块职责划分的方式。Model、View、Controller，三个模块担负着各自独立的功能。

- Model：模型，它负责对整个软件项目数据和业务的封装和管理。
- View：视图，它负责给用户传递信息，收集用户提交的信息。
- Controller：控制器，它负责控制视图的展示逻辑。

应用软件系统的最终目的都是给用户使用，MVC 中的 View 在这里就专门负责与用户交互，所有给用户展示的信息和接收用户录入信息的任务都由它来完成。但是它的任务也仅限于此，它不对要展示给用户的数据和用户录入的数据做任何处理。

那么，它的上级是谁呢？也就是说他由谁调遣呢？答案就是 Controller。View 收集的用户录入信息都得乖乖地交给相应的 Controller，由 Controller 处理完以后，把处理给果交给 View，View 就得乖乖地把数据展示给用户。

看到这里，或许你会觉得 Controller 好强大啊！但是，在现实中任何一件事物都不可能一家独大的，软件环境里也一样如此，Controller 似乎很牛气，但是它也得有牛气的后台啊，没资本它是牛不起来的。在这里它的后台就是 Model——低调的王者。

再打个比方，近几年来常关注新闻的人总多多少少会看到国务院新闻办公室召开的一些新闻发布会，那些新闻发言人谈笑风生，让人羡慕不已。

其实这些人就是 MVC 里的 View，他的上级在开会前都已经交待他们了：见记者你们得怎么怎么说，说错了不行。他们的上级就是新闻办公室(Controller)，当然，新闻办公室要发布新闻，还得忙忙呵呵地找它的后台——国务院(Model)，商量着什么能说，什么不能说，什么敢说，什么不敢说。商量完了，再回去自己整理一下，找个时间开新闻发布会，找个新闻发言人(View)在会上发布一下。

不知道这个比喻怎么样，反正我们可以直白地将 MVC 理解成一个 UI 层的设计模式。

下面来看 MVC 各模块之间的依赖关系，如图 8-1 所示。

图 8-1　MVC各模块之间的依赖关系

图 8-1 很简单明了地让我们知道了在使用 MVC 模式的系统里，用户和 View 打交道，View 请求 Controller，Controller 调用相应的 Model 处理业务逻辑，Model 根据实际需要操作数据库或资源文件，Controller 获得 Model 的业务执行结果，将结果传递给 View，由 View 对用户进行展示。

絮叨了好几遍，那么 MVC 这样做有什么好处呢？

- 低耦合性：模块的划分使得功能模块之间耦合性降低，最大的特点是视图(View)和控制器(Controller)的分离。这样允许修改 View 层的展示而不必理会控制器和模型层的代码。
- 代码可重用性：假设有一个 Web 系统，某一天又想发布 Wap 版本的系统，我们完全可以重用模型层，只开发控制器和视图就行了。
- 方便多人分工：这样能让有较强 UI 设计能力的人专注于 View 层开发，有较强业务能力的人专注于业务模型的开发。各尽其才，各尽其职。
- 易于维护和扩展：因为其极低的耦合性，所以添加和修改模块功能是都十分方便。

网上常有人拿 MVC 和普通三层结构做比较，观点有点乱七八糟，有人竟然敢从字面意思去理解、比较它们。要比较它们，得从它们的设计思想来比，也就是说当初为什么要提出这种方式来划分项目模块。

不管是 MVC 还是一般的三层架构，都是为了解决实际软件开发过程中遇到的问题的。所以从解决的问题的角度来比较是最为合适的。

MVC 主要是把页面视图、页面展示逻辑互相剥离开来。这明显地是一个处理 UI 逻辑的模式。

而常见的三层架构，一般分为表示层、业务逻辑层、数据访问层。这明显地是从整个程序结构上的分层。

需要着重说明的是：表示层的责任就是处理用户界面(UI)的，刚好和 MVC 模式所解决的问题一样，只是 MVC 更加模块化，更加专业地解决了表现层的问题。当然，从 MVC 模式的角度讲，三层架构数据访问层，业务逻辑层和实体模型层就都包含在 MVC 模式的 Model 里了，刚好，三层架构又是对 MVC 的 Model 进行的更模块化、更合理的划分。

所以 MVC 和三层架构二者没有直接的关系，同时使用也完全不冲突。



也有人把 MVC 直接叫作三层架构，其实也并不错，毕竟它就是分了三个界限明显的模块来减小耦合，也可以这么理解。不过可能会与真正意义上的三层架构在概念上有点冲突。

2. ASP.NET MVC框架

上面费了这么长的笔墨讲的都是 MVC 设计模式，下面我们来看看 ASP.NET 里的 MVC 框架。

ASP.NET MVC 框架是微软官方提供的依据 MVC 模式编写的应用于 ASP.NET Web 应用程序的一个框架。

本来可能 ASP.NET 里没想要 MVC，ASP.NET WebForm 是微软研究的相当独特的一种 Web 视图层解决方案，直接类似于 WinForm 程序一样拖拉控件，很强势地解决了程序员在 Web 开发中页面展示代码难以编写的问题，提高了页面展示的开发效率。

但任何事情都有利有弊，控件式的编程模型封装得太多，导致程序开发的自由度降低，以至于在开发某些特殊问题的时候，开发过程变得十分复杂。而且程序生成的代码毕竟不如人一样能刻意地注意一些人性化方面的问题。比如 HTML 布局的混乱导致查看和分析 HTML 代码时变得十分困难。

当然，嗅觉灵敏的微软公司也意识到了这个问题，但是独有的 WebForm 设计是举世无双的专利性产品，在这些问题上没有成功案例可以参考，他们必需独自处理和解决这些问题，但是当前所面临的挑战因为种种原因很难在短时间得到内解决。眼看市场份额在被慢慢蚕食，作为无奈之举，微软又只好引入当下发展得如火如荼的 MVC 模式，研究和开发出了具有重大意义的 ASP.NET MVC 框架。

下面就开始我们的 ASP.NET MVC 框架之旅。

8.1.2 实例描述

话说某资深程序员下班后闲来无事要学习书法，并决定在这方面一定要有所建树。于是乎他花重金购买了上等的文房四宝，还购置了一套返古风格的家具，买了名人字画，把书房收拾得文气十足。

一日傍晚突生雅兴，一番磨墨拟纸，并点了上好的檀香，颇有王羲之风范，又具颜真卿气势，气定神闲，挥毫泼墨，洋洋洒洒地写下一行大字：“Hello World”……

看完这则笑话，是不是觉得颇有程序员之风？

作为“业务”熟练的程序员，我们在每开始学一门新语言、学一门新技术的时候都会写下第一个最简单的实例“Hello World”来跟陌生的世界打一个友好的招呼。下面这个例子就是要我们“磨墨拟纸”，跟 MVC 框架打一声招呼“Hello MVC”。

这个实例比较简单——配置和调试 ASP.NET MVC 环境。对于这个问题，只要百度一下，就会有一大堆方法。不过在作者第一次用 VS2008 配置的时候还真是出了点问题，折腾了好久。可能是作者资质愚钝吧！不过问题还是解决了，下面就向读者介绍一下方法。

8.1.3 实例应用

【例 8-1】我的第一个 MVC 项目。

首先假设读者已经装好了 Microsoft Visual Studio 2008 中文版，并成功地打上 VS2008 SP1

补丁。具体的安装步骤这里不再多说。



如果你电脑上装的是 Microsoft Visual Studio 2010，那么恭喜你，你可以跳过这讨厌的安装过程了。因为 Microsoft Visual Studio 2010 已经集成了 ASP.NET MVC 2 框架。

启动 VS2008，选择“帮助”菜单下的“关于 Microsoft Visual Studio”命令。弹出的对话框如图 8-2 所示。

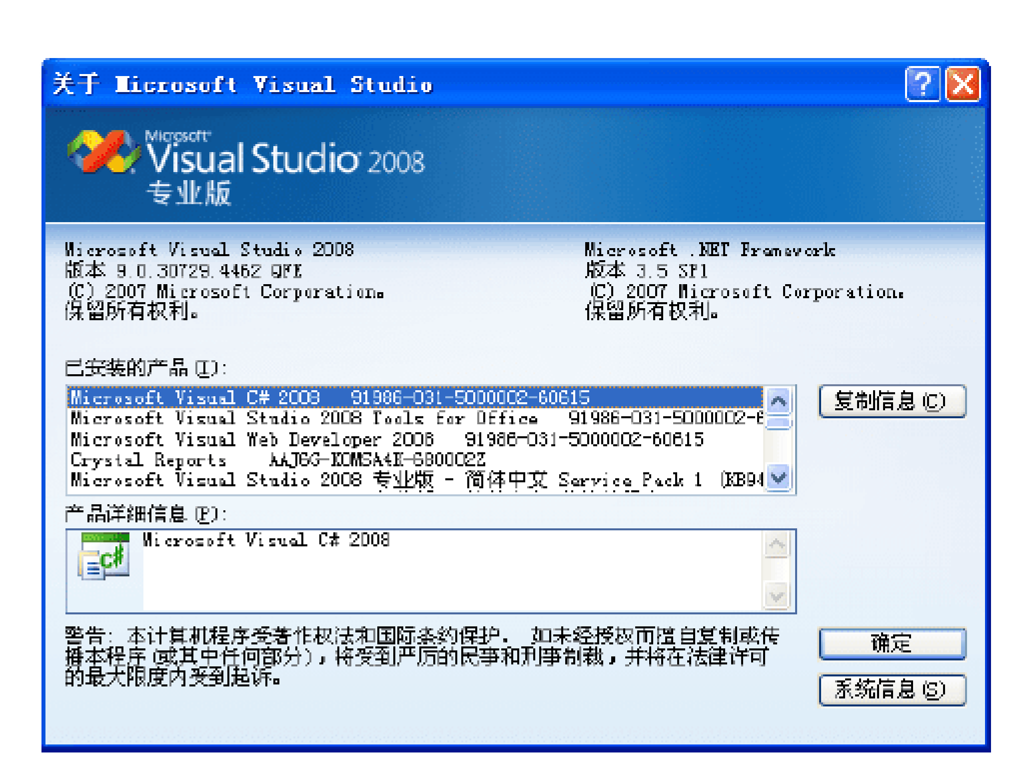


图 8-2　关于Microsoft Visual Studio

接下来需要下载 MVC 框架的安装包 AspNetMVC2_VS2008.exe，具体的安装路径这里就不写了，只要百度一下，就可以找到很多线索。



这里不是非要强调软件版本，因为所找的教程是用 AspNetMVCPreview2-setup.msi 讲的例子，安装后却报.NET 3.5 的一个类找不着。没办法，只好卸掉，然后找到这个版本装上，就可以了。

下载完以后，运行它。与一般的软件安装方法一样，同意协议(还是英文的)，下一步，坐着等。数秒后，安装成功了。

打开“控制面板”里的“添加或删除程序”，我们可以看到里面多了下面图中的两项，就说明安装成功了，如图 8-3 所示。

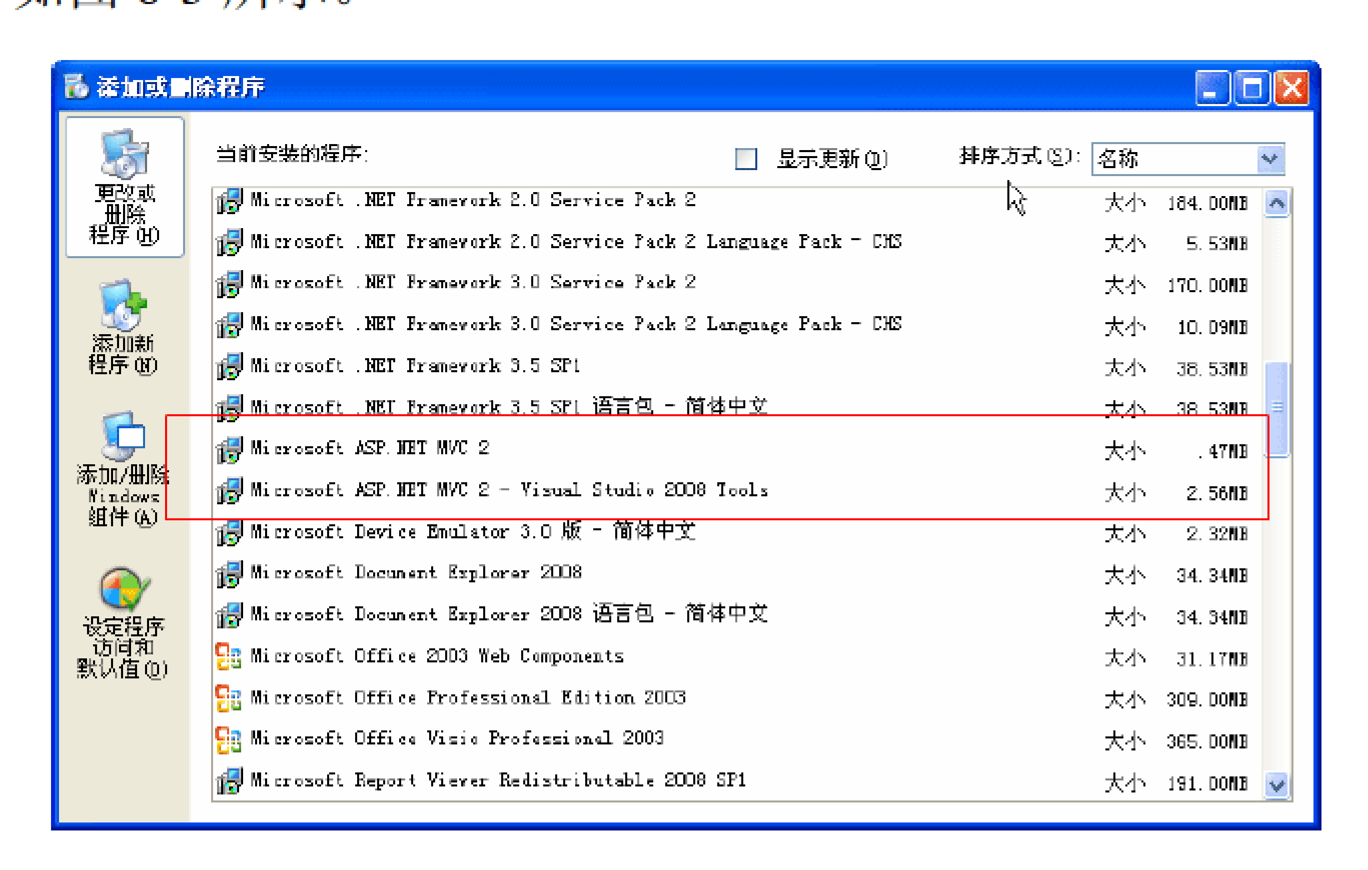


图 8-3　添加或删除程序

启动 VS2008，新建项目，怎么找不到 MVC 的模板项？百度了一下，说是因为语言版本问题，VS2008 我们用的是中文的，MVC 安装的是英文的，MVC 的模板文件在英文目录下放着，而 VS2008 要去中文的目录下找模版，所以就找不着 MVC 了。

找着原因了就很简单了，执行以下操作就行了。

(1) 打开目录 C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ItemTemplates\CSharp\Web\MVC\，将子目录 1033 重命名为 2052。

(2) 打开目录 C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ItemTemplates Cache\CSharp\Web\MVC\，将子目录 1033 重命名为 2052。

(3) 打开目录 C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates\CSharp\Web\，将子目录 1033 下的所有内容剪切到子目录 2052 下面。

(4) 打开目录 C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\ProjectTemplates Cache\ CSharp\Web\，将子目录 1033 下的所有内容剪切到子目录 2052 下面。

(5) 重建 VS2008 的模板索引，依次选择“开始”→“程序”→“Microsoft Visual Studio 2008”→“Visual Studio Tools”→“Visual Studio 2008 命令提示”。在打开的“Visual Studio 2008 命令提示”窗口中输入 devenv -setup，然后按 Enter 键。数秒以后，即可完工。



作者的 VS2008 安装在 C 盘的 Program Files 目录下，如果读者安装在其他分区或其他目录下，到安装的目录找行了。

再运行 VS2008，新建项目，项目类型选择 Visual C#下面的 Web，就会发现已经多出了两项 MVC 的项目模板，如图 8-4 所示。

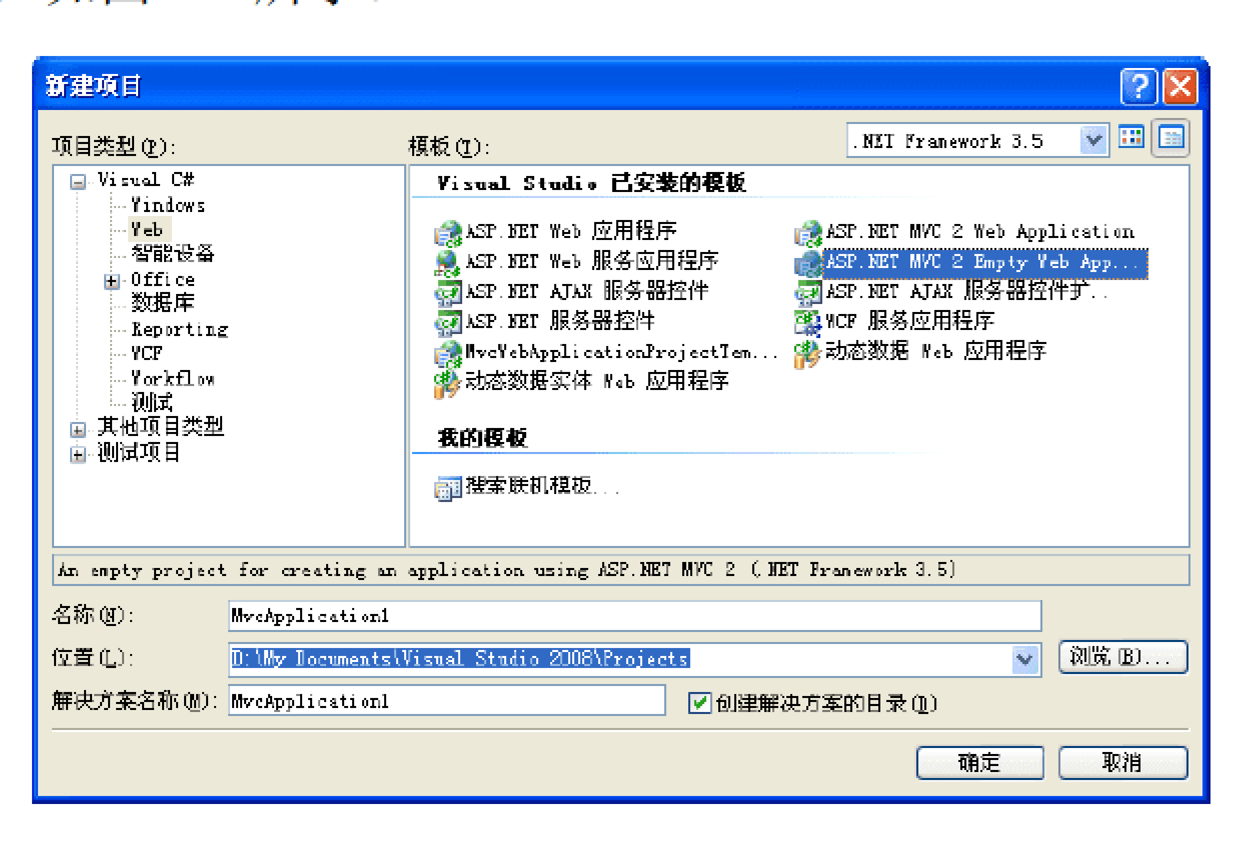


图 8-4 “新建项目”对话框

至此说明 Microsoft Visual Studio 2008 中文版下的 MVC 2 正式安装配置成功。



当然安装方法也因为各版本不同而不同，这里讲的安装方法只提供一个参考，读者可以尝试安装和使用其他更新一点的版本。

安装完毕我们来建一个项目测试一下。

刚才的“新建项目”对话框的“模版”选择框中有两个关于 MVC 的选项。

- ASP.NET MVC 2 Web Application：该选项是微软创建的一个基于 MVC 的小实例，有兴趣的读者可以新建一个进去看看，学习一下。

- ASP.NET MVC 2 Empty WebApp…：该选项能建立一个空的 MVC 项目，这个项目系统自动为我们搭建好了 MVC 环境，做了一些基本的配置。

作者不喜欢被“污染”过的程序，在这里我们选择 ASP.NET MVC 2 Empty WebApp…这一项，建立一个空的 MVC 项目，输入我们的项目名称“MyMVC Project”，选择路径到常用的项目目录里面，单击“确定”按钮。

来看一下“解决方案资源管理器”里系统为我们搭建好的目录，如图 8-5 所示。

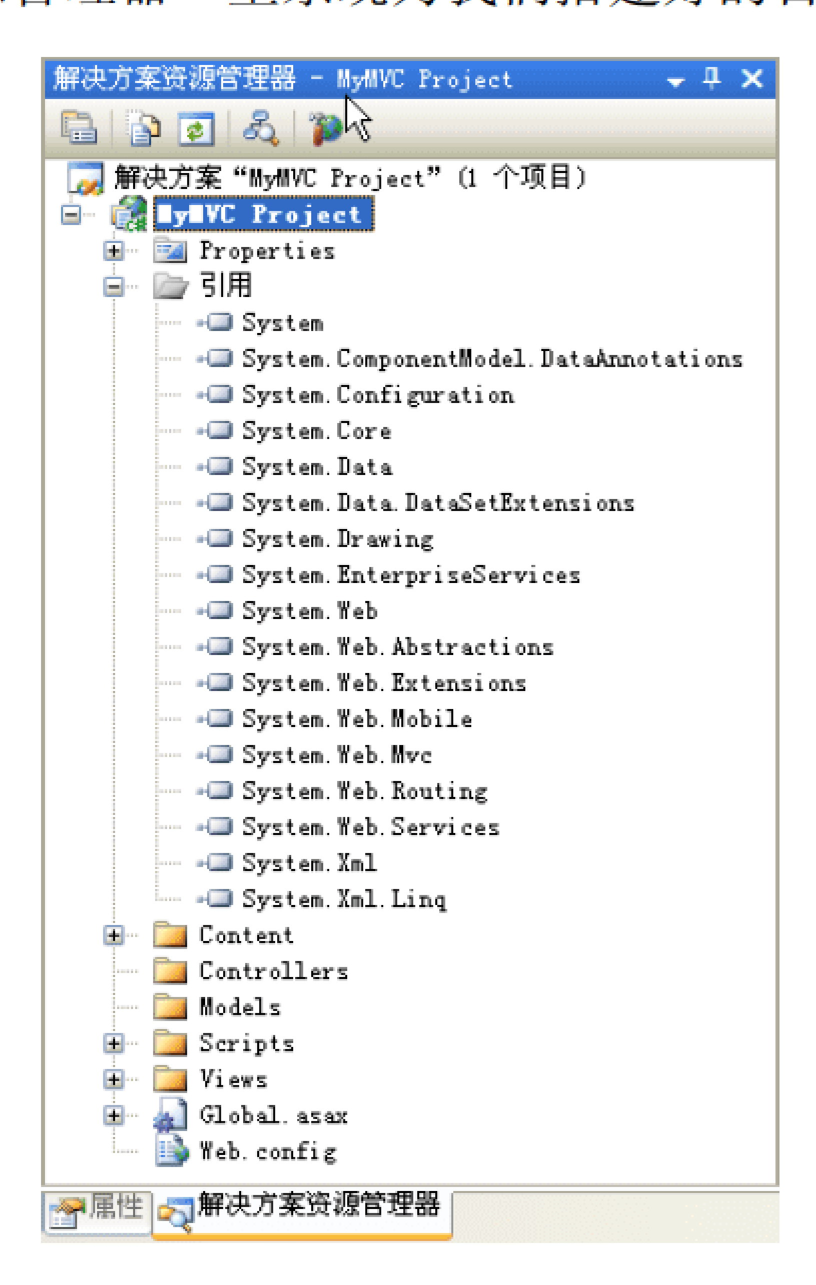


图 8-5　解决方案资源管理器

可以看到，虽然是空的 MVC 项目，但目录结构比普通的 Web 项目复杂了许多。我们来一一介绍。

- “引用”节点：首先“引用”节点下面多了几个我们以前没见过的命名空间，比如 System.Web.Mvc。这里不做深入研究，就不管它了，单击“引用”前面的带减号的小方块，把“引用”节点折叠起来。
- Content 目录：展开后发现其下面有一个样式表文件。不用猜，肯定是存放样式表之类的资源文件的。
- Controllers 目录：看目录名称就知道它是存放 Controller 的。微软建议组织好你项目中的 Controller。当然，也可以存到别的地方。
- Models 目录：目录名称也已经很明确地告诉我们，这里是放 Model 的。但真正大点的项目，都是分好多层的。比如采用三层架构的项目，数据模型、业务逻辑、数据访问都是独立的项目，谁有空把数据都弄到这里面来呢。
- Scripts 目录：放脚本文件的。展开看看，其中一大堆 JS 文件。
- Views 目录：放 View(视图)的。也就是说项目里所有的 View(视图)文件都得放到这里面统一管理。
- Global.asax 文件：用于配置访问规则。

- Web.config 文件：打开看看，比普通的 Web 项目多了不少乱七八糟的东西，也不知道有哪些作用，先不管它。

目录结构似乎没什么，但这么简单的一些东西，就已经完美地搭建好了一个 MVC 项目结构。只需要编写自己的 Model、View 和 Controller 就能制做一个标准的 ASP.NET MVC Web 项目，当然，如果程序逻辑简单，又不要求代码规范的话，甚至连 Model 都不用要(比如我们这次做的实例)。

在开始之前，既然了解过了目录结构，那再来了解一下 ASP.NET MVC 的执行流程。

当浏览器请求 Web 服务器的时候，Web 服务器接收请求并调用相应的 Web 应用程序。在这个环节，ASP.NET Framework 会根据配置文件的配置将请求拦截，交给 MVC 框架，MVC 框架根据 Global.asax 文件配置的 URL 映射规则调用相应的 Controller(视图)的 Action(动作)，Action(动作)处理完请求以后，调用相应的 View(视图)生成结果反馈给浏览器。



这里是 Action(动作)“反馈给浏览器”，而不是 View(视图)。也就是说即使没有 View(视图)，Action(动作)也一样会完整地处理请求。

在上面这段话里，又有两个新的概念。

- URL 映射规则：首先我们知道 MVC 里是由 Controller 来处理请求的，URL 路径和 Controller 的对应关系就是 URL 映射规则。
- Action(动作)：在 ASP.NET MVC 里，Controller 是以类的形式存在的，而 Action 就是 Controller 里的 public 方法。

这样，就一切豁然开朗了。接下来就可以创建我们自己的 Controller 和 View 来完成这个项目了。

首先创建一个 Controller。

我们把 Controller 放在 Controllers 目录下面，右击 Controllers 目录，再单击“添加”，然后选择“新建项”等等……

“新建项”上面有什么东西？Controller？微软的东西真的很强大，为了方便，都已经做成菜单项了，省得选“新建项”以后再选这个选那个才能创建，如图 8-6 所示。

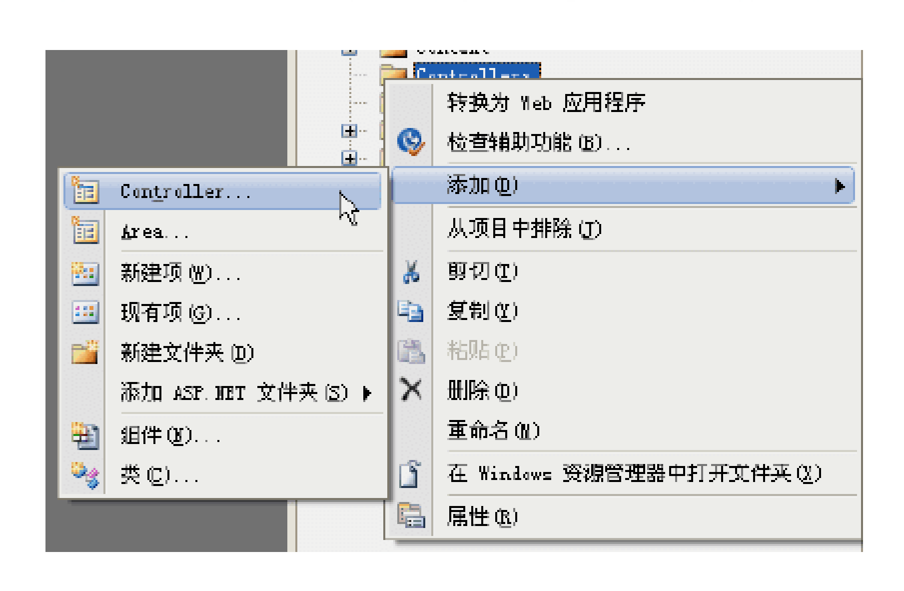


图 8-6　新建Controller菜单命令

单击 Controller，打开一个对话框，似乎是让输入名称。但它只选中前面部分是什么意思呢？查资料后才知道，原来所有的 Controller 名称必须以 Controller 结尾。

输入我们的 Controller 名称“Home”，最后文本框里完整的名称是 HomeController。接下来有一个多选框“Add action methods for…”，不使用它，不用它生成任何方法。然后单击 Add 按钮。



就像普通的 Web 项目默认的页面是 Default.aspx 文件一样，ASP.NET MVC 默认的 Controller 名称为 Home，默认的 Action 名称是 Index。当然可以在 Global.ascx 文件里配置。

这样就创建好了一个 Controller，非常简单，代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MyMVC_Project.Controllers
{
    public class HomeController : Controller
    {
        //
        // GET: /Home/

        public ActionResult Index()
        {
            return View();
        }
    }
}
```

命名空间是我们的项目名称和目录名称，类名是我们刚才输入的名称，继承了 Controller 父类，并且自动为我们创建好了默认方法 Index()。可以看到该方法的返回值是 ActionResult 类型的，直译就是“动作结果”，没有其他代码，只是返回时执行了一个 View()方法，这好神秘。

既然 Controller 建好了，我们先来运行一下试试效果，反正肯定运行不了，因为没有 View。我们就试试没有 View 是什么效果吧。按 F5 键，询问是不是调试，选“是”。接着弹出浏览器窗口，阿弥陀佛，我们想要的结果出来了：

```
异常详细信息: System.InvalidOperationException: The view 'Index' or its master
was not found. The following locations were searched:
~/Views/Home/Index.aspx
~/Views/Home/Index.ascx
~/Views/Shared/Index.aspx
~/Views/Shared/Index.ascx
```

多么经典直白的异常信息，让我们一下子明白了三个道理：

- 每个动作 Action 执行的时候默认就去找 Views 目录下的以自己所在 Controller 名称(不带后缀 Controller)命名的目录下的同名 View。
- 视图可以有两种格式——.aspx 和.ascx，而且.aspx 文件优先。
- 当在对应目录下找不到所要的视图时，就会去一个名为 Shared 的目录下找。

看到如此一针见血的异常信息，应该再次叹服一下微软产品的人性化。

从错误信息里我们知道，原来它在苦苦寻寻觅着一个叫 Index 的视图文件。知道了就好说

了，那就创建吧。

上面说过，ASP.NET MVC 框架要求 View 放在 Views 目录下面。我们找到 Views 目录，展开后发现里面已经有一个名为 Shared 的目录和一个 Web.config 文件。Web.config 文件就不说了，肯定是对这个目录进行配置的。再看那个目录，不就是上面异常信息里提到的 Shared 目录吗？百度一下，果然如它的名字所起的那样，这个目录里存放一些共享的 View 文件，如果 Controller 在自己对应的目录里找不着对应的视图文件，就到这个共享目录里找，再找不着的话就报上面那个异常了。

好了，在 Views 目录下创建名为 Home 的目录，对应我们的 HomeController 控制器。右击 Home 目录，单击“添加”菜单项，又出现了类似添加 Controller 时的那个菜单，不过这次菜单项 Controller 变成 View 了。真方便！

单击 View 菜单项。弹出一个 Add View 对话框。输入名称，这里输入 Index，与 Controller 里的 Action 对应。还不能创建？看下面选项。

- Create a partial view(.ascx)：意思是创建一个局部视图，从后面的扩展名可以知道，选中这项创建的是一个用户控件。这项我们不管它。
- Create a strongly-typed view：意思是创建一个强类型的视图，我们不管它。
- Select master page：选择母版页，我们没有。所以取消选中。

这样就可以创建了，单击 Add 按钮。

费尽周折，终于创建成功了。这时我们的 Views 目录的 Home 目录下应该有一个 Index.aspx 文件，打开它。看源代码发现它除了 Page 指令不一样了，body 里没有 form 表单了，其他的和普通 ASP 页面的源代码一个样。我们来重新运行项目。

这次没有报异常，而是出现一个空白的页面。这说明程序已经调试通过了。

接着我们来向页面中写点东西试试。在 body 标签里默认创建的 div 中加入一行代码：

```
<h2>Hello MVC!!!</h2>
```

大功告成！

8.1.4 运行结果

直接运行项目，浏览器上打印出“Hello MVC!!!”，如图 8-7 所示。

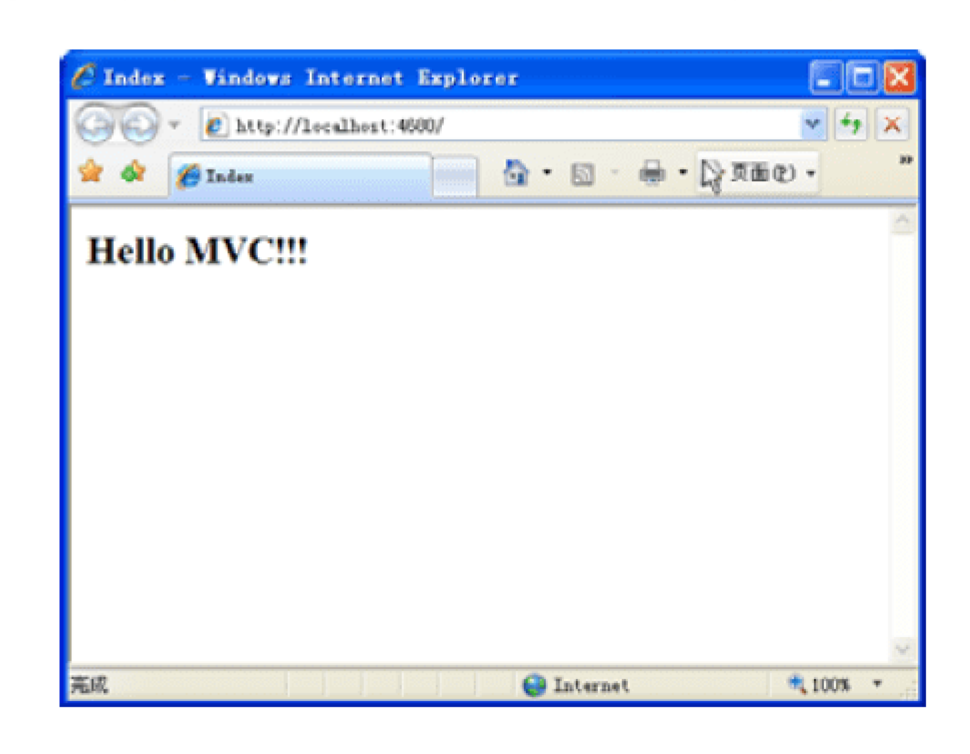


图 8-7　程序运行结果

8.1.5　实例分析



源码解析：

本实例主要来了解一下 ASP.NET 下的 MVC 框架，以及 MVC 框架的安装过程。其实安装过程并无特殊之处，只是因为版本语言问题而需要简单地配置。

安装完以后我们创建了一个简单的 MVC 项目了解了 MVC 项目的目录结构，以及项目的创建和执行流程。

8.2　实现有自己特色的URL路径

没事在网上闲逛，总会进一些网站，有时候见一个好的贴子或者好的博文，经常想记下来以后慢慢看。但抬头看看它那乱糟糟的 URL，简直让人崩溃。随便举个例子：

```
http://www.*****.com/search_blog.aspx?nf=2009&yf=3&rq=5&kw=mvc
```

这个时候，我们绝对没法记在脑子里，不得不把它复制保存到其他地方，下次想起来的时候还得去上次保存的地方找，非常麻烦。

如何做到更为简短又有意义呢？

ASP.NET MVC 已经给我们提供了一个非常优秀的解决方案，答案就是 UrlRouting。

它能把 URL 简化成这个样子：

```
http://www.*****.com/Blog/Search/2009/3/5/mvc
```

干净，明了。是不是非常先进呢？

接下来我们就来了解一下。



视频教学：光盘/videos/8/URLRouting.avi　　　　**长度：**18 分钟

8.2.1　基础知识——URLRouting

URLRouting 是让设计者根据自己的需要制定一些 URL 规则，用于解析用户请求到实际的地址。

比如我们知道在 ASP.NET MVC 里 Controller 实际上是一个类，我们想让这个类处理用户请求，需要将浏览器请求根据请求地址解析到某一个具体的方法，这里我们就要用到 URLRouting。

使用 URL routing，需要规定 URL 模式，它包括一个位置标识，将在你请求网页时按这个规则返回内容。当然，这个创建的规则完全是由你自己定义的。

在 ASP.NET MVC 里，这些规则就定义在 Global.asax 文件里。

先来看一下新建项目时的初始 Global.asax 文件内容：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;
using System.Web.Routing;

namespace MyMVC Project
{
    // Note: For instructions on enabling IIS6 or IIS7 classic mode,
    // visit http://go.microsoft.com/?LinkId=9394801

    public class MvcApplication : System.Web.HttpApplication
    {
        public static void RegisterRoutes(RouteCollection routes)
        {
            routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

            routes.MapRoute(
                "Default", // Route name
                "{controller}/{action}/{id}", // URL with parameters
                new { controller = "Home", action = "Index",
                id = UrlParameter.Optional } // Parameter defaults
            );
        }

        protected void Application Start()
        {
            AreaRegistration.RegisterAllAreas();

            RegisterRoutes(RouteTable.Routes);
        }
    }
}
```

整个代码结构就不说了，我们来看类的内容。

类里面有一个方法 Application_Start()，顾名思义，ASP.NET MVC 项目每次运行都先执行这个方法。

这个方法在运行时注册 Route，所有的路由规则都存放在 RouteCollection 里，作为 RouteTable 的 Routes 属性供程序调用。在程序初始化的时候需要先初始化这个 RouteCollection，这里调用了一个静态方法 RegisterRoutes()来初始化 RouteCollection。

我们来看这个静态方法，默认的情况下里面只有两条完整的语句。第一句的方法名是 IgnoreRoute，第二句的方法名是 MapRoute，光看名字就知道第一句是定义要忽略的路由，第二句定义了一个映射路由。

MapRoute()方法有 3 个参数，第一个是路由名称，这里用的是 Default，说明是默认路由，系统最初只访问域名没有输入路径时，执行这个路由规则；第二个参数是路由规则，也就是路

由路径格式，这里定义的是“控制器/动作/ID(参数)”；第三个参数是该路由规则的默认值。这里默认的 Controller 是 Home，默认 Action 是 Index，当然，参数 id 是可选项。这也就是前面我们创建实例时为什么要以 Home 命名的原因。

8.2.2　实例描述

有时候上网我们会见到一些很特殊的 URL，这个 URL 就像在访问一个目录一样，并没有类似网页文件的扩展名，也没有随着问号引出的一串参数，但是它执行了和平常的动态网站一样的功能。但看起来 URL 更加简洁，也容易记忆了，听说还更利于搜索引擎优化。

比如：

```
http://www.*****.com/Blog/Search/2009/3/5/mvc
```

这样很容易就能知道是要搜索 2009 年 3 月 5 号的博客，后边的 MVC 很可能就是要搜索的关键字。

这样的 URL 更短，更加容易记忆，也更加好看了。这是怎么实现的呢，我们接下来通过实例来研究一下。

8.2.3　实例应用

【例 8-2】实现有自己特色的 URL 路径。

打开上一节创建的项目 MyMVC Project。我们来模拟一下搜索指定日期和关键字的博客的 Route 的配置，只在页面显示一下参数信息，就不做访问数据库的操作了。

根据实例需求，需要创建一个名为 Blog 的 Controller，里面有一个名为 Search 的 Action，该 Action 有 4 个参数：year、month、day、keyword，分别接收年月日和关键字等参数。具体代码如下：

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Mvc;

namespace MyMVC_Project.Controllers
{
   public class BlogController : Controller
   {
      //
      // GET: /Blog/

      public ActionResult Search(int year, int month, int day, string keyword)
      {
         string message = "您要搜索" + year + "年" + month + "月"
            + day + "日发表的关于“" + keyword + "”的文章。";
```

```
            ViewData["Message"] = message;
            return View();
        }
    }
}
```



代码中用到了数据呈现字典 ViewDate，下节会详细介绍，这里不做过多说明。我们只须知道这是一个类似 Dictionary 的集合，可以存放任何对象，并能在 View 直接访问，是 Controller 和 View 传递数据的媒介。

这里我们用不到名为 Index 的 Action，为了方便，就把它删掉了。这里能把提交的数据当方法参数传递，这也是 ASP.NET MVC 的一大特色。

接着我们创建一个 View。当然还得先创建一个与 Controller 同名的目录，在该目录下创建一个名为 Search 的视图。并在视图文件 body 部分的 div 里加如下一行代码：

```
<h2><%= ViewData["Message"] %></h2>
```

这样就可以访问到前面在 Controller 里存放的字符串对象了。

接着我们修改 Global.asax 文件的映射 Route 部分。在默认的 Route 前面添加一条 Route。最终的 RegisterRoutes()方法如下：

```
public static void RegisterRoutes(RouteCollection routes)
{
    routes.IgnoreRoute("{resource}.axd/{*pathInfo}");

    routes.MapRoute(
        "BlogSearch",
        "{controller}/{action}/{year}/{month}/{day}/{keyword}",
        new { year = 2010, month = 1, day = 1, keyword = "mvc"}
    );

    routes.MapRoute(
        "Default", // Route name
        "{controller}/{action}/{id}", // URL with parameters
        new { controller = "Home", action = "Index", id = UrlParameter.Optional }
        // Parameter defaults
    );
}
```



这里一定要将添加的 Route 添加到默认 Route 前面。因为系统是从上到下去匹配第一个满足条件的 Route。如果你放在后面，肯定会被匹配更为广义的默认 Route 先拦截走。

这里我们新添加了一个 Route 配置 BlogSearch，有 4 个参数。它将映射具有参数列表 year，month，day，keyword 的 Action 请求。就是说请求的目标 Action 最多需要这 4 个对应名称的参数，如果方法参数在该 Route 里没有出现，则不被映射。

比如下面这个 Action 也会被该 Route 映射：

```
public ActionResult About(int year, int month, int day)
{
    return View();
}
```

我们运行一下，访问这个路径：

```
http://localhost:4680/Blog/Search/2009/3/5/mvc
```

是不是想要的结果都出来了？

另外，我们加一点特色，Route 可以这样写：

```
routes.MapRoute(
    "BlogSearch",
    "{controller}.mvc/{action}/{year}/{month}/{day}/{keyword}",
    new { year = 2010, month = 1, day = 1, keyword = "mvc"}
);
```

也就是在{Controller}后面加上“.mvc”，这样我们访问这个路径的时候，Controller 就需要加上“.mvc”访问。路径如下：

```
http://localhost:4680/Blog.mvc/Search/2009/3/5/mvc
```

当然，Action 也可以这样加上一些后缀，让 URL 拥有自己的特色。

8.2.4　运行结果

运行项目，访问的还是默认的/Home/Index，我们手动在址栏中输入/Blog.mvc/Search/2009/3/5/mvc，就可以看到运行结果，如图 8-8 所示。

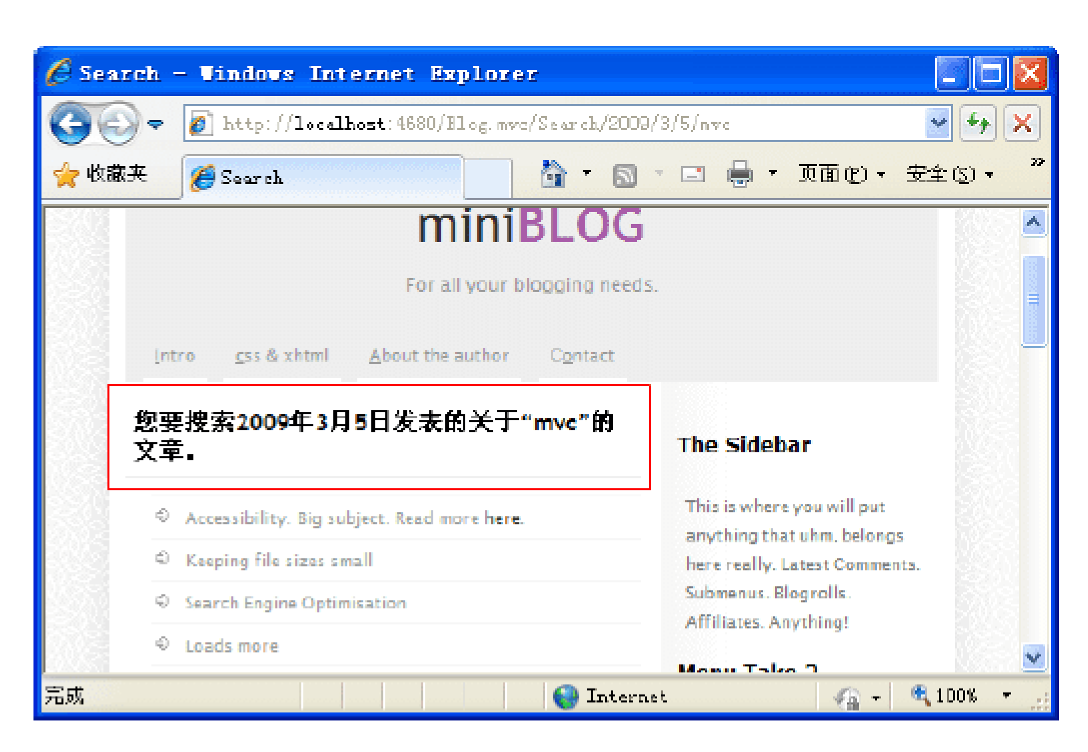


图 8-8　运行结果

8.2.5　实例分析



源码解析：

这个实例主要熟练一下 Route 的配置，其实主要就是 MapRoute()方法的使用。还有一点是

最重要的，就是了解 MVC，再深入体会一下 ASP.NET MVC 框架的设计思想和工作流程。

实际的 URLRouting 要讲的东西很多，限于篇幅原因，这里只说了最常用的配置。有兴趣的读者可以自己研究。

8.3 显示服务器信息

前面我们已经简单地了解了一下 MVC 程序的运行机制。我们知道，为了降低耦合，MVC 三个模块间的关系十分独立。

Controller 负责数据的读取，而 View 负责界面的呈现，并且在界面的呈现过程中 View 通常不进行数据的读取和逻辑运算(虽然它可以，但是尽量避免)，数据的读取和逻辑运算都需要交给 Controller 负责。

那么它们之间是如何通信的呢？我们知道 Controller 和 View 是完全独立的，没有一点直接关系，而不像 ASP.NET Web Form 那样页面和页面后台代码之间可以相互引用。那么 Controller 在处理完数据以后，是怎样把它交给 View 进行展示的呢？

通常我们对某一个访问者的数据进行暂存时，总习惯使用 Session 对象。在这里也可以，但是随之而来的问题是：在 Session 里保存的临时数据还要我们编写繁琐的代码进行管理，否则它会大量地浪费服务器内存，影响应用程序的性能。

当然，以“优雅”著称的.NET，在这个问题上也给我们提供了很好的解决方案，那就是 ViewData 和 TempData。

其实上一节我们已经使用过了 ViewData，没有做详细的解释，下面我们来研究一下。

 **视频教学：光盘/videos/8/ViewData&TempData.avi　　长度：8 分钟**

8.3.1 基础知识——ViewData和TempData

有人会问，同一个问题，为什么 ASP.NET MVC 需要提供两个解决方案呢？

其实它们虽然说都能实现数据传递，但还是有些差别的。先从名字来说：ViewData 是数据呈现字典，TempData 是临时数据呈现字典(也有人说是跨页数据呈现字典)。ViewData 只能在当次请求时被访问，TempData 可以在设置完以后同一会话的任何一次请求被访问，但只限一次请求。

比如我们只在 Action A 里同时设置了一个 ViewData[“Key”]和一个 TempData[“Key”]：

我们访问 Action A 返回的是 View A，

接着又访问了 Action B，返回 View B，

接着又访问了 Action C，返回 View C，

接着又访问了 Action D，返回 View D，

……

在以上的执行过程中，每一个 ABCD 编号就是一次请求。因为我们只在 Action A 里对它

们存值，所以我们能且只能在 Action A 和 View A 里访问 ViewData[“Key”]里保存的值，在其他三次请求里我们不能访问到 ViewData[“Key”]的值。

但是 TempData[“Key”]有点不太一样，TempData 可以被理解为是一个一次性的 Session，也就是说存进去的值，可以在会话的生存周期内的任何一次请求中访问。所以这个例子中 TempData[“Key”]能在上面 ABCD 任何一次请求中访问，但是访问完它的值就被清空。

8.3.2　实例描述

在很多应用程序里，特别是网站后台，为了给用户更好的体验，往往会在用户登录成功后的欢迎页面里显示一些服务器的基本信息，比如主机名、IP 地址、站点根目录之类的信息，以方便让用户更清楚地知道自己的应用程序的运行环境。

本实例就借展示服务器信息的名义介绍一下数据呈现字典 ViewData 和临时数据呈现字典 TempData 的用法。

它们二者颇为相似，在这里提醒读者多体会一下二者的差别。

8.3.3　实例应用

【例 8-3】显示服务器信息。

打开前两节使用的项目 MyMVC Project。我们来用数据呈现字典，向 View 呈现一些信息。

至此我们已经对 MVC 比较熟悉了，那种“Hello World”深情的招呼在这里也显得有点没有必要了。所以这里我们就拿/Home/Index 修改一下，完成我们今天的例子。

首先修改 HomeController 的 Index 方法为如下效果：

```
public ActionResult Index()
{
    string ServerName = Dns.GetHostName();      //获取服务器名称
    IPAddress addr = Dns.GetHostAddresses(ServerName)[0];
    string ServerIP = addr.ToString();          //获取服务器 IP
    string AppPath = Server.MapPath("~/");      //获取应用程序目录

    //将信息存入 ViewData
    ViewData["ServerName"] = ServerName;
    ViewData["ServerIP"] = ServerIP;
    ViewData["AppPath"] = AppPath;

    //将信息存入 TempData
    TempData["ServerName"] = ServerName;
    TempData["ServerIP"] = ServerIP;
    TempData["AppPath"] = AppPath;

    return View();
}
```



这里用到了两个网络操作类 Dns 和 IPAddress，所以需要引入.NET Framework 的网络操作类所在的命名空间 System.Net。

再修改 Home 目录里的 Index.aspx 视图的 body 部分为如下效果：

```
<div>
<h2>Index</h2>
<h3>ViewData</h3>
<p style="color:Red;">
ServerName: <%= ViewData["ServerName"]%><br />
ServerIP: <%= ViewData["ServerIP"]%><br />
AppPath: <%= ViewData["AppPath"]%><br />
</p>
<h3>TempData</h3>
<p style="color:Blue;">
ServerName: <%= TempData["ServerName"]%><br />
ServerIP: <%= TempData["ServerIP"]%><br />
AppPath: <%= TempData["AppPath"]%><br />
</p>
</div>
```

这里看代码就可以知道，分别以红色和蓝色显示 ViewData 和 TempData 里存储的内容。来看一下运行结果，如图 8-9 所示。

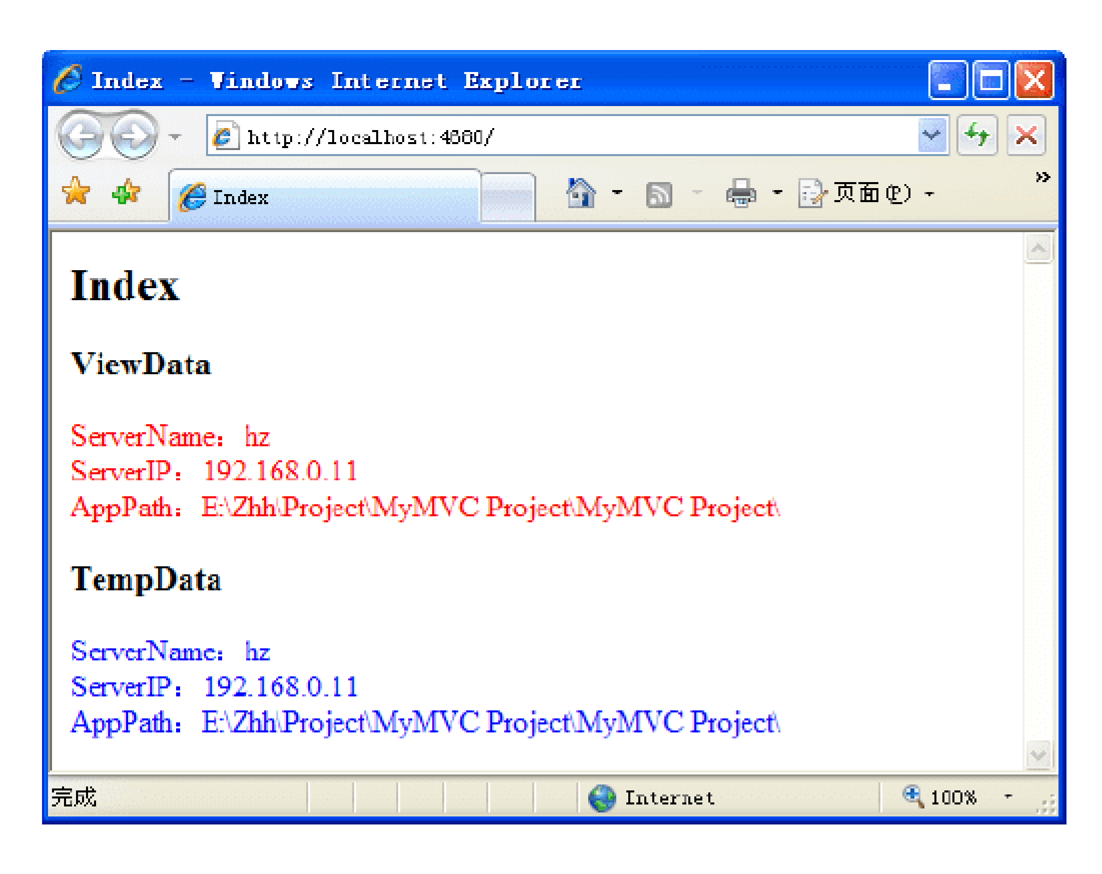


图 8-9　数据呈现字典测试运行结果(一)

可以看到，它们两个都成功地传递了数据。

似乎没有什么区别啊。接下来再做一下测试。

我们在 HomeController 里创建两个 Action，一个名为 Set，内容和 Index 一样，设置一下值；另一个名为 Show，内容只有一行 return View("Index");，意思是直接返回名字为 Index 的 View，不做任何数据设置，只用于显示数据。

同时还要在/Views/Home 目录下创建一个名为 Set 的视图文件，内容不做修改。

现在，我们先来访问一下/Home/Set，没有报错，显示空白页面。当然，这属于正常，是 Set 视图。接着，我们再来访问一下/Home/Show，富于创意的结果出现了。ViewData 结果一个没有，TempData 的数据安然无恙。如图 8-10 所示。

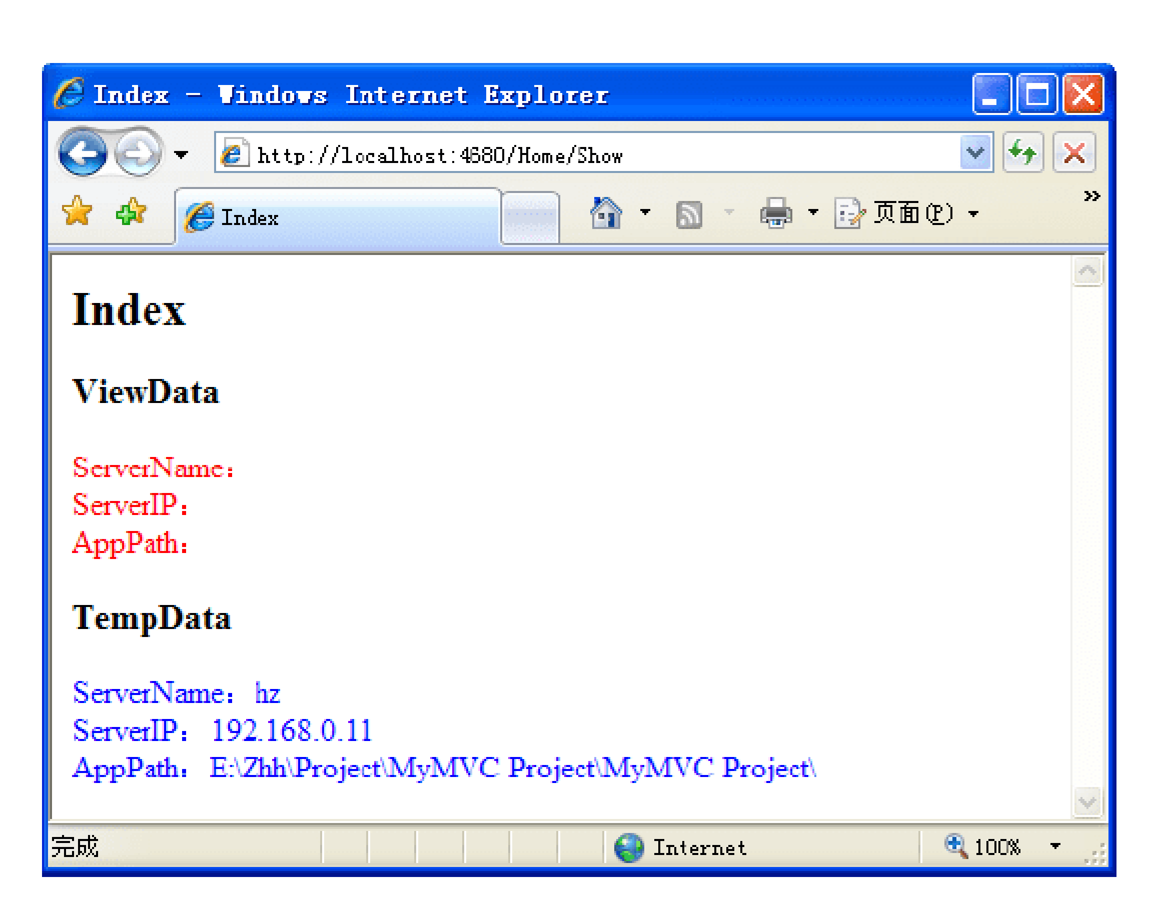


图 8-10　数据呈现字典测试运行结果(二)

这似乎预示着什么，访问了两次，TempData 正常显示……

可能有些读者已经想到了，TempData 被称为跨页数据呈现字典，数据可以比较持久地在服务器内存中保存，我们可以在会话生存周期内的任何时间访问，它都可以使用。

但是，对它一次设置只能被某“一次”请求访问。注意这里啊，只能被一次请求访问。绝对不容质疑，能多次访问它就成 Session 了。

不信就来试试。

我们先访问/Home/Index，在这次请求里，我们既对数据呈现对象做了设置，又在 View 中取出了它们的值，结果两组数据都能显示(参考图 8-9)。

接下来，我们再来访问/Home/Show。

结果如图 8-11 所示，证明了上述结论的正确性。

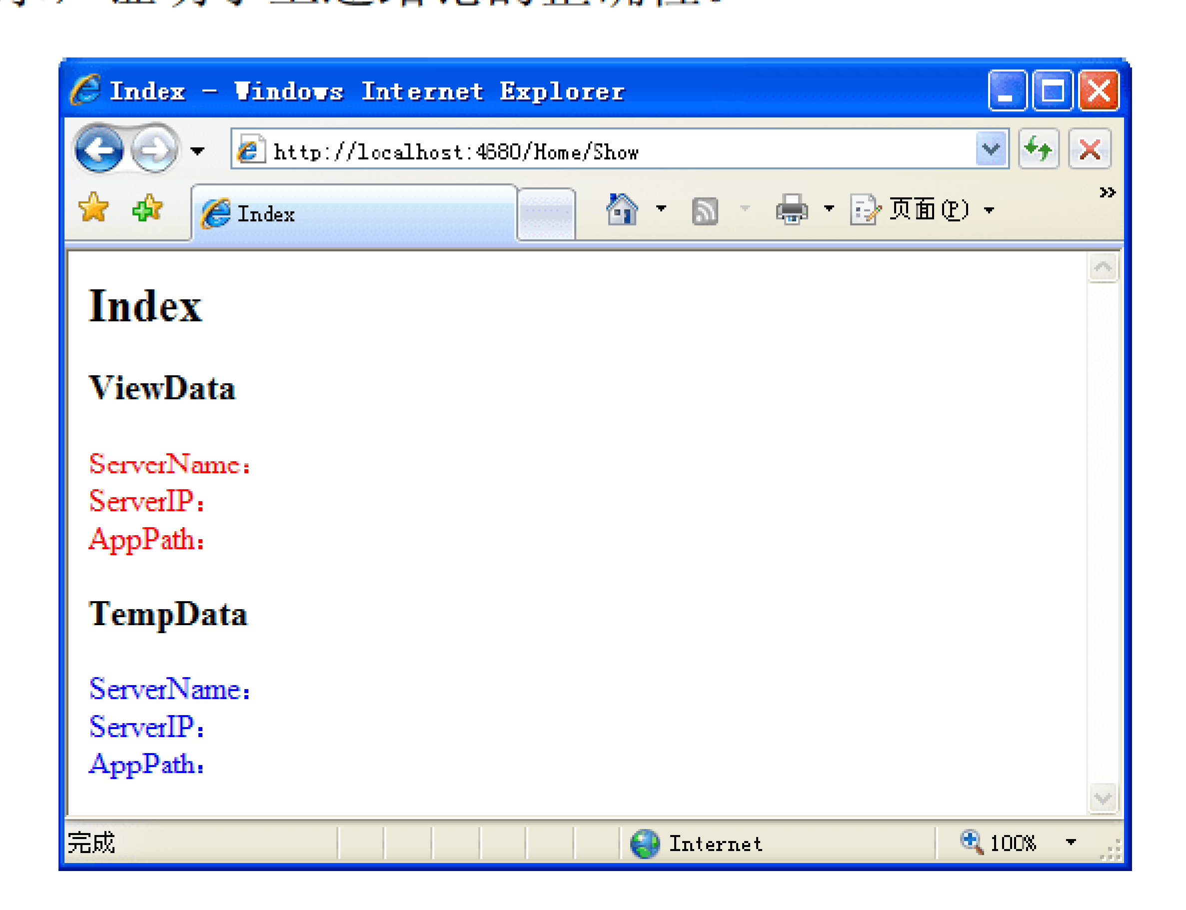


图 8-11　数据呈现字典运行结果(三)

我们在访问/Home/Index 时所有的数据都显示过了，所以这里一个都不显示了。当然 ViewData 并不只是已经访问过一次的原因，它本身就不能跨页。



这个实例我们共做了三组测试，因为浏览器缓存的原因，可能显示结果事与愿违，所以建议每做一组测试新打开一个浏览器窗口。

8.3.4 实例分析



源码解析：

从这个实例我们了解了 ViewData 和 TempData 两个数据呈现字典的用法和区别。还需要说明的一点是，它们存储的都是 object 类型的数据，也就是说它们能存储任何对象。用的时候在 View 里执行一下转型操作就可以了。

这里使用了 View()方法的另一个重载 return View("Index");，这样就能够返回一个名为 Index 的视图。当然还有更多的用法，不再多解释了。

8.4 基于MVC的用户登录

改用 MVC，看到要复古的页面视图，心里总感觉十分别扭。

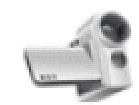
还用以前的 ASP 程序里常用的纯 HTML 标签？对于拖习惯了 ASP.NET WebForm 的控件的我们来说有点太痛苦了吧，一点都不爽。

但再不爽也不能用 Web 控件啊。一用就乱了，就又回到 WebForm 开发模式了。

用过 JSP Struts 的读者都知道，Struts 下面有一套 JSTL 标签，上手快，用着也相当顺手。

ASP.NET MVC 呢？一向人性化和傻瓜式的微软产品，这个问题上不会做得太差劲吧！

果然不错，ASP.NET MVC 也提供了一套页面控件辅助类——HtmlHelper。



视频教学：光盘/videos/8/HtmlHelper.avi　　长度：9 分钟

8.4.1 基础知识——HtmlHelper

HtmlHelper 提供了一系列的方法，可以让用户很方便地生成相应的 HTML 代码，大大地简化了编码人员的工作。

常用的方法如表 8-1 所示。

表 8-1　HtmlHelper方法

方 法 名	说　明
BeginForm	开始一个表单
EndForm	结束一个表单
CheckBox	生成一个多选框
Hidden	生成一个隐藏域
Password	生成一个密码框
RadioButton	生成一个单选按钮

续表

方 法 名	说　明
TextBox	生成一个文本框
ActionLink	生成一个超链接
RouteLink	根据路由信息生成一个超链接
DropDownList	生成一个下接列表框
ListBox	生成一个列表框

为了让开发人员更方便地使用 HtmlHelper 控件，ASP.NET MVC 的设计人员在视图 ViewPage 类中设置了一个 Html 属性，代码如下：

```
Public HtmlHelper Html{ get; set; };
```

从上面的代码我们可以看出，Html 属性是一个 HtmlHelper 类型的对象。因此我们在视图中使用 Html 对象就可以调用实现控件的相关方法。

1. BeginForm和EndForm

这两个方法是为了生成一个完整的 Form 表单，一个头，一个尾。尾部的 EndForm 就不说了，它就生成一个</ form>。

BeginForm 重载了十几种方法。我们了解一下最常用的几种。

例如：

```
Html.BeginForm();
```

这种方式生成一个采用 Post 方式提交到本页面的 Form。在默认页面/Home/Index 生成的代码如下：

```
<form action="/" method="post">
```

当然，这是默认页面，如果你在其他页面，这里的 action 属性就是那次请求的 URL 路径了。再来介绍一下另一种常用的方式：

```
Html.BeginForm(object routeValue);
```

这样根据提供的 routeValue 匿名类，根据 URLRouting 生成一个匹配的 URL 的 Form。

例如：

```
Html.BeginForm(new { controller="Home", action="PostForm"});
```

生成：

```
<form action="/Home/PostForm" method="post">
```

还有一种作者习惯用的方式：

```
Html.BeginForm(string actionName, string controllerName);
```

这个直接指定要生成的 URL 的 Action 和 Controller。

例如：

```
Html.BeginForm("PostForm", "Home");
```

一样生成：

```
<form action="/Home/PostForm" method="post">
```

还有一种，与刚才讲的第三种差不多，只写一个 actionName，生成当前 Controller 里的指定 Action。

2. 文本录入控件TextBox和Password

这两个控件使用更加方便。

它们的使用方法一样，下面我们来介绍一下常用的三种重载方法。

第一种：

```
Html.TextBox(string name);
```

生成一个指定 name 属性的文本框。

例如：

```
<%= Html.TextBox("loginName")%>
```

生成：

```
<input id="loginName" name="loginName" type="text" />
```

第二种：

```
Html.TextBox(string name, object value);
```

生成一个指定 name 属性和默认值的文本框。

例如：

```
<%= Html.TextBox("loginName", "admin")%>
```

生成：

```
<input id="loginName" name="loginName" type="text" value="admin"/>
```

第三种：

```
Html.TextBox(string name, object value, object htmlAttributes);
```

生成一个指定 name 属性，指定默认值，并且具有匿名类中指定属性的文本框。

例如：

```
<%= Html.TextBox("loginName", "", new{ size=20 })%>
```

生成：

```
<input id="loginName" name="loginName" type="text" size="20" value=""/>
```

这样就介绍完毕。比较简单，也很容易理解。密码框的用法和文本框一模一样，只是在前台显示数据不一样，这里不再重复解释。

下面来看实例。

8.4.2　实例描述

任何动态网站都要用到信息管理功能。当然管理信息就需要有适当的权限，否则若任何人都能登录并修改信息的话，整个系统就乱套了。

权限认证方式有很多，但是使用账号密码登录的认证方式以其独有的方便性一直占据主流，一时间还难以有其他什么方法能与其一争高下。

本实例中，我们借实现用户登录功能来学习一下 HtmlHelper 页面辅助工具。

在这里，我们重点讲解 ASP.NET MVC 框架，就不再进行数据库操作了，只是简单地模拟一下数据验证功能。

8.4.3　实例应用

【例 8-4】基于 MVC 的用户登录。

前面已经对我们最初创建的项目改得不成样子了，这次我们创建一个新的项目 MyMVC。

这个实例中，我们需要对数据进行封装，还需要处理简单的验证业务逻辑，所以我们需要使用 Model。

首先在 Models 目录下面创建一个存放 Model 的 cs 文件，内容如下：

```
using System;

namespace MyMVC.Models
{
    public class UserInfo
    {
        /* 用户信息实体 */
        public string LoginName { get; set; }   //登录名
        public string Password { get; set; }    //密码
        public string Username { get; set; }    //姓名
    }

    public class UserManager
    {
        /* 用户业务实体 */
        /// <summary>
        /// 验证登录名和密码
        /// </summary>
        public static bool Validate(string loginName, string password)
        {
            //这里模拟一下用户验证，实际项目中需要读取数据库用户信息进行验证
            return "zhang" == loginName && "123" == password;
        }

        /// <summary>
```

```
        /// 根据登录名获取用户信息
        /// </summary>
        public static UserInfo GetUserByLoginName(string loginName)
        {
            //这里模拟一下数据库查询
            return new UserInfo()
            {
                LoginName = "zhang", Password = "123", Username = "张学良"
            };
        }
    }
}
```

这样 Model 就完工了。接下来我们创建 Controller。

先在 Controllers 目录下创建一个名为 Account 的 Controller，Index 方法默认，再创建一个 PostForm 方法，用于接收用户登录的提交操作。PostForm 方法的代码如下：

```
public ActionResult PostForm()
{
    string loginName = Request.Form["loginName"];
    string password = Request.Form["password"];

    if (Models.UserManager.Validate(loginName, password))
    {
        Session["CurrentUser"] =
          Models.UserManager.GetUserByLoginName(loginName);
        return Redirect("/Account/Success");
    }

    ViewData["LoginName"] = loginName;
    ViewData["ReturnMessage"] = "用户名或者密码不正确";
    return View("Index");
}
```

这里我们首先获取用户提交的登录名和密码，调用相应 Model 的验证方法验证，如果验证成功，记录当前用户信息，跳转到登录成功页面。

如果验证不成功，向视图发送提示信息，返回登录视图。细心的读者可能注意到了，我们不但向视图发送了一个提示信息，还向视图发送了一个登录名称。这样做有什么作用呢？一会再揭开谜底。

上面说了，如果验证成功，跳转到验证成功页面，当然，还得有一个 Action 用来处理成功请求。

所以我们再在 Account 控制器里创建一个名为 Success 的方法，方法结构参考 Index。

Controller 至此完工，接下来就是 View 了。

上面的例子中一共有 3 个 Action，但是 PostForm 只是处理了一下提交请求，并没有做任何呈现功能。所以我们只需要一个 Index 视图作为登录页面，一个 Success 视图作为登录成功页面。

首先来看一下 Index 视图的主要代码：

```
<div>
<% Html.BeginForm("PostForm","Account"); %>
    登录名：<%= Html.TextBox("loginName",ViewData["LoginName"]) %><br />
    密　码：<%= Html.Password("password") %><br />
    <input type="submit" value="Submit" />
    <%= ViewData["ReturnMessage"] %>
<% Html.EndForm(); %>
</div>
```

似乎猛一看有点陌生，不过看看这个，再比比纯 HTML 代码，还是觉得这个好看点。其实不光好看，它们功能强大得很，以后慢慢研究。这里我们看到第 3 行与其他有点不太一样，后面多一个参数，那是设置默认值的。这里将 Controller 传递过来的数据呈现到文本框里。

当然，这样设计好处多多，比如用户登录的时候可能是不小心密码输入不正确，返回的时候，当然自动填上这个，会人性化不少的。

可能只从一个用户名我们看不出来能有多大便利性，但是，假如要注册或者录入其他繁多的信息时，这样就会显得非常重要。接下来看一下 Success 视图的主要代码：

```
<div>
<% MyMVC.Models.UserInfo currentUser =
     Session["CurrentUser"] as MyMVC.Models.UserInfo; %>
你好，欢迎你：<%= currentUser.Username %><br />
你的登录名称是：<%= currentUser.LoginName %>
</div>
```

因为我们登录成功以后将用户信息保存到了会话当中，表明当前用户已经登录。所以，我们在这里从会话中取出用户信息，并显示出来。当然，Session 对象存储的对象都是 object 类型的，所以取出来的时候需要对它进行转型，然后再输出。

8.4.4　运行结果

编译，运行项目。访问/Account/Index。显示结果如图 8-12 所示。

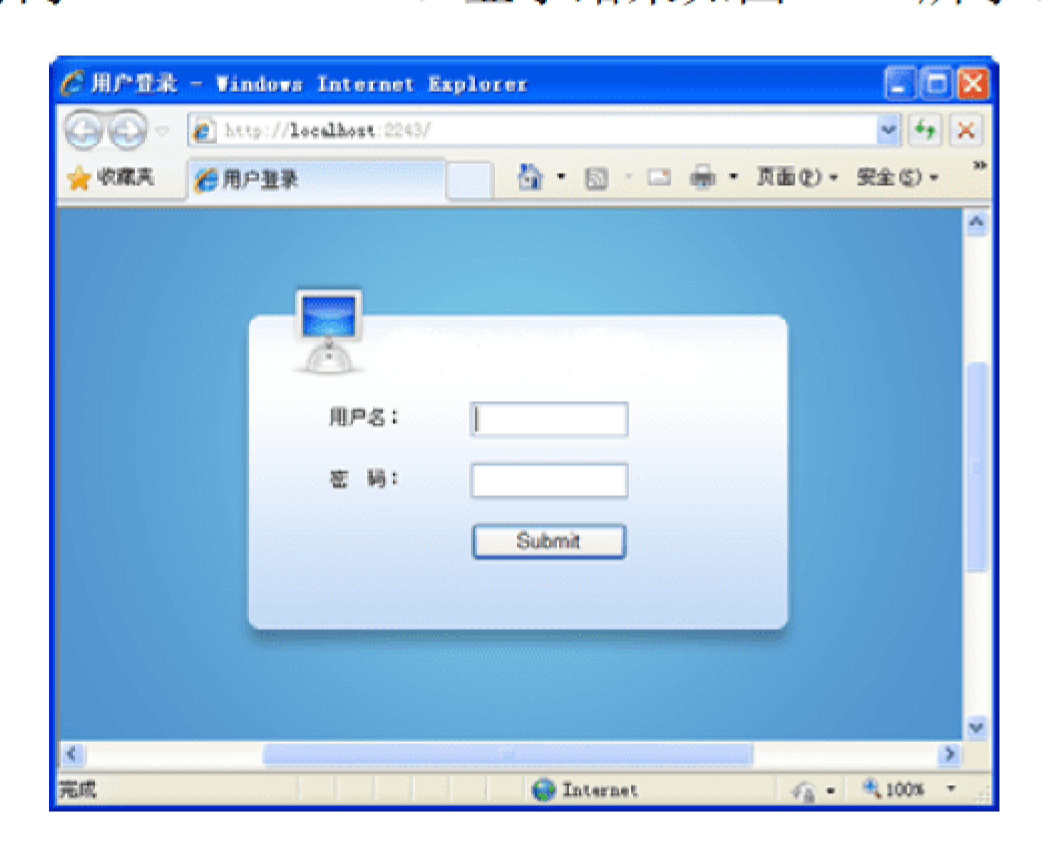


图 8-12　用户登录界面

在这里我们输入错误的用户名和密码，系统会在 Submit 按钮后面提示“用户名或者密码不正确”，如图 8-13 所示。



图 8-13　用户登录失败

我们再输入正确的用户名 zhang 和密码 123，单击 Submit 按钮。结果如图 8-14 所示。



图 8-14　用户登录成功

8.4.5　实例分析

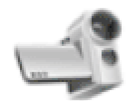

源码解析：

这一节由于篇幅的原因，没有对 HtmlHelper 控件进行一一讲解。用完以后是不是觉得这样对于我们程序员来说，相对更容易理解，更容易掌握呢。另外，BeginForm 方法还有另外一种用法：using(Html.BeginForm(“ActionName”, “ControllerName”)){}。用花括号括住表单内容，这种方法更为简洁，而且不用 EndForm 方法结尾了。代码是不是更漂亮了？另外，还有一些 URLHelper 与这个类似，有兴趣的读者自己研究一下。

8.5　使用Filter过滤用户查看信息操作

软件系统中危机重重，每一次请求都可能带有不安全因素。如何降低危险性，有很多解决方案。要想办法对每次请求进行检查(保护每一个 Action)。当然 ASP.NET MVC 也提供了相应的安保措施，这就是 Filter(过滤器)。

下面我们就来了解一下。


视频教学： 光盘/videos/8/Filter.avi　　　　长度：7 分钟

8.5.1　基础知识——Filter

我们知道在 ASP.NET MVC 里，浏览器的请求被映射到相应的 Controller 中的 Action，并由 Action 执行完以后返回结果给浏览器，如图 8-15 所示。

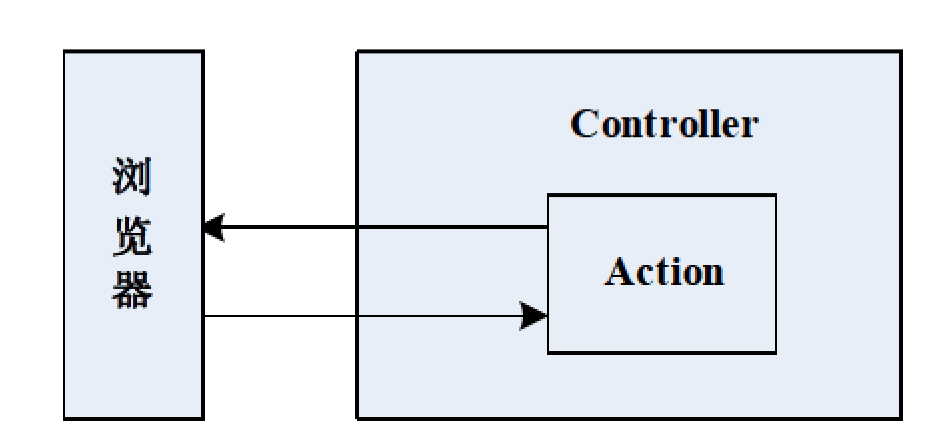


图 8-15　用户请求执行流程图

当然，这个时候浏览器可能被用户操纵，请求带有一些危险的操作。那么 Action 必死无疑。假如在 Action 里写大量的验证代码。Action 安全了，也显得不伦不类了，有点违背“单一职责”的原则了。

所以，ASP.NET MVC 中引入了一个 Filter 机制，来保证在 Action 不被修改的情况下提供 Action 的安全监控功能。

ASP.NET MVC 里的 Filter 可以有两种，一种是 Controller 的过滤器，它对 Controller 下的所有 Action 提供过滤功能，一种是 Action 的过滤器，它只对选定的某几个 Action 进行过滤。

如果把它们都应用到软件项目中，流程如图 8-16 所示。

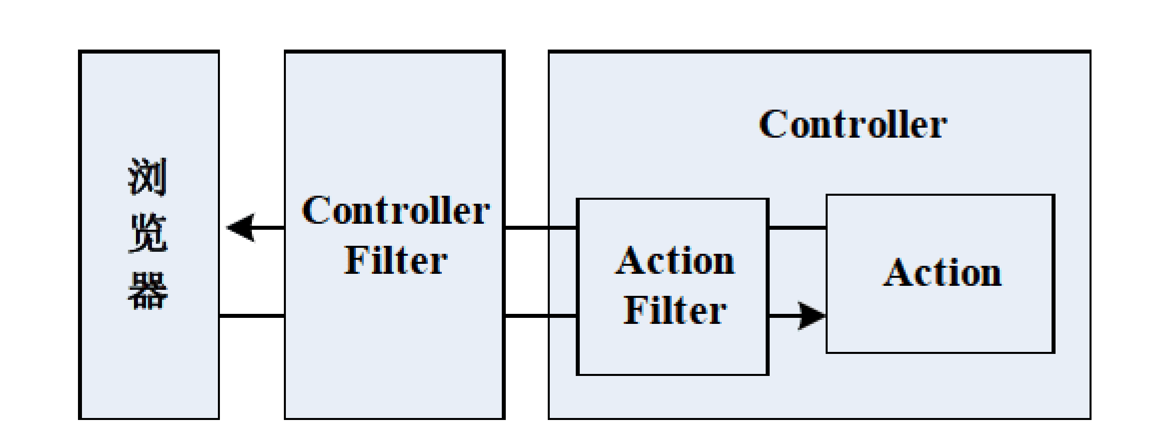


图 8-16　Filter过滤用户请求流程图

从图 8-16 中我们可以很明白地看到，两类 Filter 可以对请求做两次过滤。非常全面地对 Action 的安保机制提供了支持。

8.5.2　实例描述

我们运行上面的实例，觉得它运行得完美么？能保证它的安全么？

作者感觉是不能的。

比如我们重新打开一个浏览器窗口，并直接访问/Account/Success。怎么样，出错了吧！如图 8-17 所示。

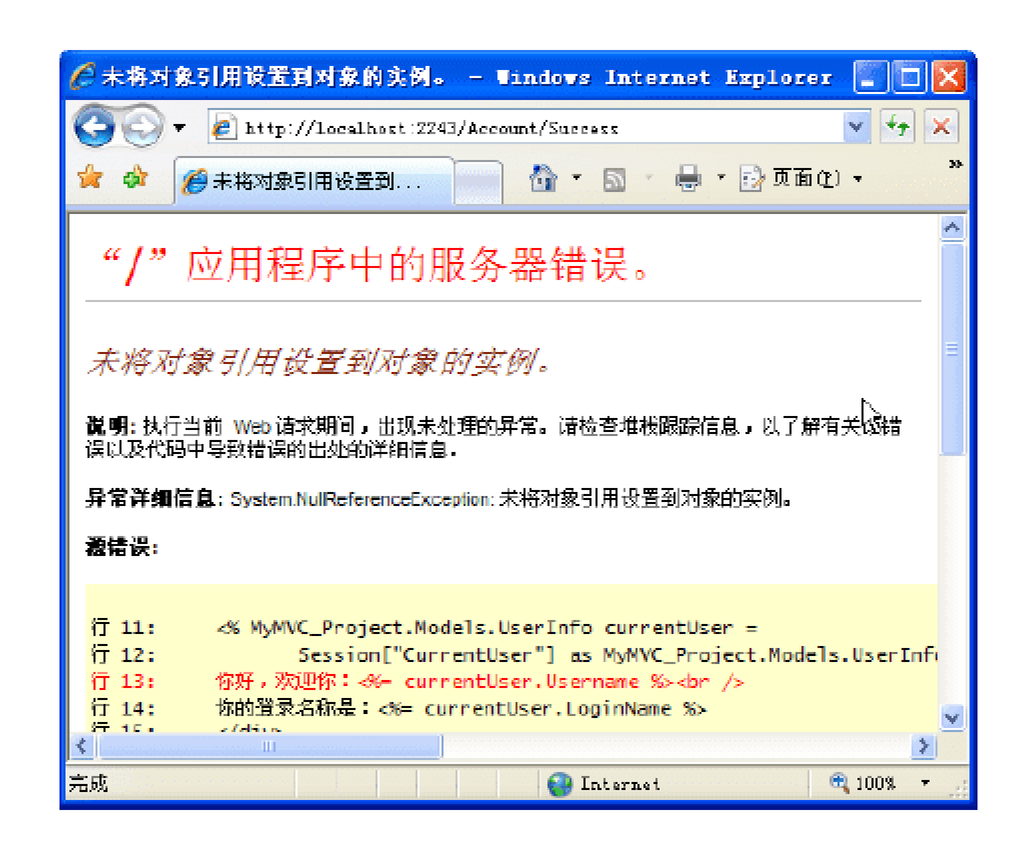


图 8-17　直接访问/Account/Success的出错信息

从错误提示上可以看到，当前用户为 null，直接访问它的属性，当然就错了。

有的读者可能会想，我们在这里加一个判断不就行了？的确可以。但是你有没有察觉到你又要去修改臃肿的 Action 甚至是 View 的代码了。

要避免这样做，可以给此 Action 加一个 Filter(过滤器)。Filter 应继承于 ActionFilterAttribute 类，并可以覆写 void OnActionExecuting(ActionExecutingContext)和 void OnActionExecuted (ActionExecutedContext filterContext)这两个方法。看名字我们就可以知道，这两个方法一个是 Action 执行前执行，一个是 Action 执行后执行。

接下来我们来看实例。

8.5.3　实例应用

【例 8-5】使用 Filter 过滤用户查看信息操作。

打开我们前面使用的项目。

为了便于管理 Filter，先在根目录下创建一个 Filter 目录，用于存放所有的 Filter。本实例要用到的 Filter 就存放在该目录下。

右击新建的 Filter 目录，选择“添加”二级菜单下面的“类”菜单项，创建一个普通的类，这里我们命名为“UserPowerFilter”。

修改这个类，让它继承 ActionFilterAttribute，当然，需要引入 System.Web.Mvc 命名空间。这里只需在执行 Action 前验证用户权限，所以重写 OnActionExecuting(ActionExecutingContext)方法，并添加代码。

整个 Filter 代码如下：

```
using System;
using System.Web.Mvc;
using System.Web.Routing;
using MyMVC.Models;

namespace MyMVC.Filter
{
    public class UserPowerFilter : ActionFilterAttribute
    {
        public override void OnActionExecuting(
          ActionExecutingContext filterContext)
        {
            base.OnActionExecuting(filterContext);
            UserInfo currentUser = filterContext.HttpContext
                .Session["CurrentUser"] as UserInfo;
            if (null == currentUser)    //如果当前用户为 null，返回登录视图
            {
                // 这句是让当前 Filter 上下文跳转到以
                // 默认路由为规则的 Account 控制器的 Index 动作上
                filterContext.Result = new RedirectToRouteResult("Default",
                    new RouteValueDictionary(
                      new { controller="Account", action = "Index" }));
            }
        }
    }
}
```

这样就建好了，但如何使用呢？

非常简单，只需要在被过滤的 Action 前面标记一下就行了。标记格式是一对方括号括住 Filter 的名字。比如这里是[MyMVC.Filter.UserPowerFilter]。

那么 Success 方法就成这个样子了：

```
[MyMVC.Filter.UserPowerFilter]
public ActionResult Success()
{
    return View();
}
```

这样就大功告成了。测试运行一下，访问/Account/Success，又回到登录页面了。神奇吧！

8.5.4　实例分析



源码解析：

使用 Filter 处理问题，很多时候会让你的程序看起来很优雅，但也注意不要滥用。前面我们提过针对 Controller 的过滤器，其实创建和使用更加方便，直接在 Coutroller 里重写 void

OnActionExecuting(ActionExecutingContext)和 void OnActionExecuted(ActionExecutedContext)方法即可，这样只要是对该 Controller 下的 Action 请求，就会执行这两个方法了。

8.6 定义全局异常处理

百密一疏，没有谁能够保证自己写的代码是完全优秀、没有一点 Bug 的。任何程序在完工的时候总还会有这样或那样的一些不足。无论是学生的毕业设计，还是公司的商业项目。

软件帝国微软开发的操作系统应该是用户最多的软件，应该算足够伟大了吧。但它任何一个版本的系统都在以每年数以百计的速度打补丁、修 Bug。

可能对软件不很了解的人会觉得有点不可思议。其实相对于生存法则、社会制度相对完善的现实世界，软件世界可以说还在远古时代，人们还在以自己对软件的认知对其进行探索。

扯了这么远，无非是想让每个程序员都低调地认识一下自己代码的不足，尽可能多地进行弥补。

既然总会有些事情让我们想象不到，我们就必须要做好“必死”的心理准备和最坏的打算。我们即使不能对所有的问题选择最优秀的解决方法，但我们还可以对可能出现的不正常现象进行收集和整理，以利于以后的维护和修缮。

下面我们就研究一下在 ASP.NET MVC 里如何对整个程序可能出现的各种异常进行统一处理。

 **视频教学**：光盘/videos/8/OnException.avi　　　　长度：9 分钟

8.6.1 基础知识——OnException

实现全局异常处理的方法很多，在 ASP.NET MVC 下面，我们就用一种 ASP.NET MVC 特有的最简单的方案来实现这个功能——重写 Controller 的 void OnException(ExceptionContext)方法。

Controller 的每一个子类都可以重写这个方法，并自己对异常信息进行处理。

该方法的原始结构如下：

```
protected override void OnException(ExceptionContext filterContext)
{
    base.OnException(filterContext);
}
```

在这个方法里我们可以使用参数 filterContext 的 Exception 属性获取当前所抛出的异常，可以进行记录或其他处理。

当然，如果每个 Controller 都重写 OnException 就有点太费力气了，而且不容易统一管理和修改。那我们可以重写一个 Controller，让系统所有的 Controller 都继承自它，一切问题就迎刃而解。

8.6.2　实例描述

前面我们使用的例子中，在执行一些操作的时候，可能会有一些不可预知的异常。

比如我们操作数据库，可能数据库会连接失败，也可能执行的时候有些细节问题我们没有考虑周全。

总之这里异常产生于我们的意识之外，虽然我们想象不到它们，但是我们知道它很可能存在。所以要收集这些信息，记录和保存，以便以后对整个系统进行分析和评估。

8.6.3　实例应用

【例 8-6】定义全局异常处理。

运行我们前面使用过的项目，在它的基础之上添加功能。

获取的异常信息需要记录日志，所以需创建一个专门记录日志的日志管理器 LogManager 类。代码如下：

```
public class LogManager
{
    string LogFilePath = null;

    public LogManager(string logFilePath)
    {
        this.LogFilePath = logFilePath;
        FileInfo file = new FileInfo(logFilePath);
        if (!file.Exists)
        {
            //这里创建以后得到一个 FileStream 对象，并处于打开状态，我们需要将它关闭
            file.Create().Close();
        }
    }

    public void SaveLog(string message, DateTime writeTime)
    {
        try
        {
            string log = writeTime.ToString() + "  " + message; //日志信息
            StreamWriter sw = new StreamWriter(LogFilePath, true);  //创建写入流
            sw.WriteLine(log);  //往流里写入一行数据
            sw.Close(); //关闭流
        }
        catch { }
    }
}
```

我们再创建一个处理异常信息的控制器 ExceptionController，其他所有控制器都继承自它。

下面是它的代码：

```
public class ExceptionController : Controller
{
    protected override void OnException(ExceptionContext filterContext)
    {
        base.OnException(filterContext);
        //初始化日志记录器
        LogManager logManager =
          new LogManager(Server.MapPath("~/") + "system.log");
        logManager.SaveLog(
          filterContext.Exception.Message, DateTime.Now); //记录日志
    }
}
```

这样我们的全局异常处理控制器就创建完成了。捕获异常以后系统会将异常信息写入系统根目录下的 system.log 文件中。

我们让所有的控制器都继承自它，它就起作用了。比如我们上节使用的 AccountController，要改为如下声明：

```
public class AccountController : ExceptionController
{
    //内容省略
}
```

至此完成了整个系统的设计。

当然，对于这个逻辑如此简单的项目，很难会跳出什么异常，我们就直接抛出一个异常来测试项目。

修改 PostForm 方法，在它被请求的时候抛出一个异常。代码如下：

```
public ActionResult PostForm()
{
    if (true)
    {
        int errorCode = new Random().Next(1000, 9999);  //随机一个四位整数
        string errorMsg = "System Not Access! Error Code:" + errorCode;
        throw new Exception(errorMsg);
    }
    ... //原来的代码省略
}
```

一切完工！来测试一下。

8.6.4 运行结果

编译运行该项目，在登录页面单击提交按钮，系统报出异常(因为我们是调试项目，所以在前台浏览器直接报出异常)，如图 8-18 所示。

图 8-18　异常测试浏览器端

我们再到系统根目录下查看日志文件的内容，如图 8-19 所示。

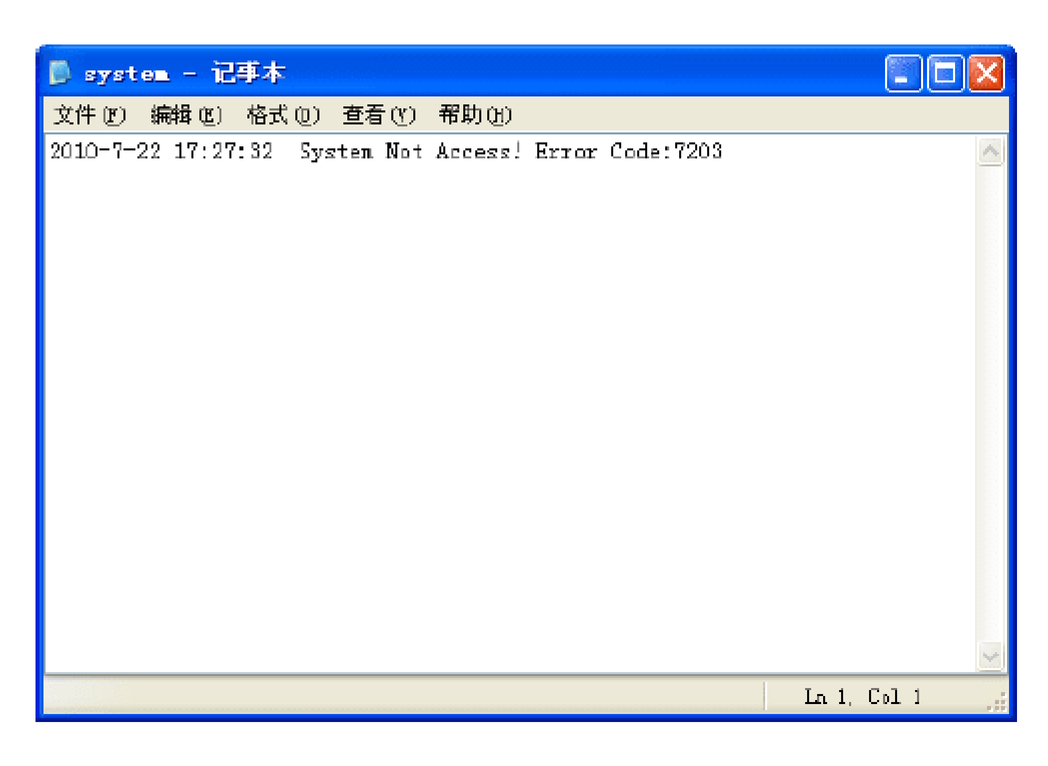


图 8-19　异常测试日志文件

8.6.5　实例分析



源码解析：

整体来说这个实例没有什么难点。只是简单地重写一下 Controller 的 void OnException (ExceptionContext)方法。使用 ExceptionContext 对象的 Exception 属性获取异常信息记录一下。这次我们操作日志文件，使用了 IO 对象，这里强调一下，任何 Stream 对象都是资源，用完时必须及时关闭，否则会影响下次使用。

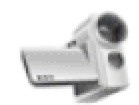
8.7　MVC里的Ajax实现

Ajax 即 Asynchronous JavaScript and XML，中文意思是“异步 JavaScript 和 XML”。它所用到的技术早已有之，近些年来随着 Web 2.0 的发展，它也跟着风靡开来。有人说它新瓶装旧

酒，这里我们不管它是新酒还是旧酒，喝着过瘾就行。

在 Ajax 推广使用前的 Web 系统中，用户执行一个操作时，整个浏览器窗口先全变白，等了好久，页面一点点、一点点地显示出来，是个很痛苦的等待过程。

Ajax 的优点就是不用刷新页面，使用户体验更具连贯性。点击了一个操作，在系统返回结果之间，页面没有任何变化。该看电影看电影，该读小说读小说，无聊了看看广告也比盯着白屏好。一旦接到系统返回结果了，就在页面该变的地方变一下，换成我们要的东西。我们还是该干嘛干嘛。基本上完全避免了那种傻乎乎等待的过程。

 **视频教学**：光盘/videos/8/MVCAjax.avi　　　　**长度**：8 分钟

8.7.1　基础知识——Ajax

当然，ASP.NET MVC 下的 Ajax 也完全颠覆了传统 ASP.NET WebForm 项目 Ajax 的使用方法。又回到古老的 JavaScript 世界。这里我们不能再拖拉控件，必须弄清客户端代码是客户端代码，服务器响应是服务器响应。

首先，我们得了解一下 Ajax 的原理。我们知道 JavaScript 完全可以在页面没有任何感觉的前提下把页面的信息改掉，用户完全没有等待感。所以用 JavaScript 是一个相当不错的选择。

当然，JavaScript 再强大也得去服务器请求数据，刚好 JavaScript 可以使用浏览器的异步数据获取技术 XMLHTTPRequest。使用 XMLHTTPRequest 技术请求完服务器获取返回结果，再用 JavaScritp 对返回的结果进行处理，展示到页面上。

所以，Ajax 技术是一个纯粹的基于 JavaScript 客户端的技术，它和具体的服务器端技术(ASP/JSP/PHP/ASP.NET)没有任何关系。

既然是基于 JavaScript 的客户端技术，它难免会因为世间千奇百怪的浏览器内核的不同而具有不同的实现方式。真是令人头疼的东西！

不过还好，在这个强人辈出的时代，早有人在我们脚边放了无数垫脚石，我们只要踩上去，就会更轻易地触及我们的梦想。

这里我们为了省去繁杂的 JavaScript 代码，使用一种 JavaScript 框架来简化开发，这就是 JQuery。

简单地了解了一下 JQuery，不得不说它真是一个绝好的东西。一些常用的和不常用的，关于浏览器兼容性的，或者费时费力的 JavaScript 操作，它都进行了封装，进行了优化。正因为它的强大，而且无私的开发团队免费提供给全球用户使用，微软在开发 ASP.NET MVC 框架时默认集成了 JQuery，使用户能够更方便地使用它。

说到 JQuery 的优点，首先提到的就是它的选择器，它竟然提供了一个类似 CSS 的 DOM 选择器，而且更为强大，更为完善，让我们简单地写一点点代码就能实现精确的对 DOM 的定位。这是作者体会最深的一点，其他的功能也非常不错，比如今天我们要用到的 Ajax。

JQuery 的 Ajax 十分简单，简直无话可说，看代码：

```
$.get("Test.aspx");
```

这样就可以了。

都不知道该怎样解释了。

不过官方给的 API 非常详细，该语句的完整格式是这样的：

```
$.get(url, [data], [callback], [type])
```

参数具体的含义如下。

- url：要请求的路径。
- data：请求时所附加的参数，格式如{ name: “John”, time: “2pm” }。
- callback：请求完毕以后被调用的处理函数。
- type：请求返回内容的格式——xml，html，script，json，text，_default。

官方的解释也同样简单，因为它实在很简单，复杂的东西已经封装完了。

下面来看实例。

8.7.2　实例描述

很多软件系统都不是单用户的，往往是每个用户一个账号，使用的时候自行登录。而且很多公开的系统都提供用户注册功能。在注册的时候就出现问题了，其他输入用 JavaScript 就可以验证合法性，用户重名怎么办？

当然，在页面提交的时候也可以进行判断。但当用户满怀欣喜地去点击 OK 按钮的时候，你去告诉它重名了。什么心情？一次可以，两次也行，三次就想哭了。

其实对于这个问题我们可以有更好的解决方案，就是在用户输完用户名要输入下一项的时候，验证一下，告诉它成功还是失败，这样谁心里都会好受点的。

本实例就演示一下用户名重名的验证。

8.7.3　实例应用

【例 8-7】MVC 里的 Ajax 实现。

我们来创建一个新的 MVC 空项目 MyMVC，当然作者是在新的解决方案里创建的。

我们展开 Scripts 目录，瞅见了吧，好些个 JQuery 的 JavaScript 文件，先不管它们都是干什么的吧。这里我们只用到了一个 jquery-1.4.1.js，其他不管它。

当然，处理请求还得有 Controller，我们就建一个默认的 Home 吧。

Index 方法不用管，加一个验证 Action，起名叫 ValidateUsername。代码如下：

```
public ActionResult ValidateUsername(string id)
{
    if ("admin" == id)
    {
        Response.Write("用户已存在!");
    }
    else
    {
        Response.Write("该用户名可以注册!");
    }
```

```
    return null;
}
```



因为 URLRouting 里映射的参数名是 id，这里我们就用 id 了，其实它就是提交过来的 Username。

我们对传过来的参数进行验证(这里只是模拟了一下)。如果存在就直接往响应结果里写一句话“用户已存在！”，如果可以注册就往响应结果里写一句话“该用户名可以注册！”。

响应这样就算处理完成了，就不再创建相应的 View 了。

好了，我们创建一个 Index 动作对应的 View。

主要代码如下：

```
<%
using (Html.BeginForm("Index", "Home"))
{
%>
    用户名：<%=Html.TextBox("Username")%><br />
    密码：<%=Html.Password("Password")%><br />
    电话：<%=Html.TextBox("Phone")%><br />
    地址：<%=Html.TextBox("Address")%><br />
    <input type="submit" value="Submit" />
<%
}
%>
```

这样就模拟了一个简单的表单，我们来添加相应的客户端事件，来实现“用户名”文本框失去焦点时到服务器验证是否存在。

接下来我们在页面的 head 部分加入如下代码：

```
<script src="../../Scripts/jquery-1.4.1.js" type="text/javascript"></script>
<script type="text/javascript">
    //默认结构，JQuery 的所有处理事件都要放在这里面
    $(function() {
        //对 name 为 Username 的文本框添加失去焦点事件处理程序
        $("input[name='Username']").blur(function() {
            //请求相应的 URL，执行相应的处理程序
            $.get('<%= Url.Action("ValidateUsername") %>/' + $(this).val(),
              function(result) {
                alert(result);
            });
        });
    });
</script>
```

只需这么几行程序就行了。

第一行引入 JavaScript 文件，不用写，直接拖着“解决方案资源管理器”里的相应文件到这里就行了。

其他的代码都有注释，不再解释。

8.7.4　运行结果

编译，运行程序，在“用户名”文本框里输入 admin，按 Tab 键切换焦点。就会弹出对话框提示“用户已存在！”，如图 8-20 所示。



图 8-20　Ajax用户名验证(一)

当你在文本框里输入其他用户名(如 joker)再按 Tab 键切换焦点时，就会弹出对话框提示“该用户名可以注册！”，如图 8-21 所示。

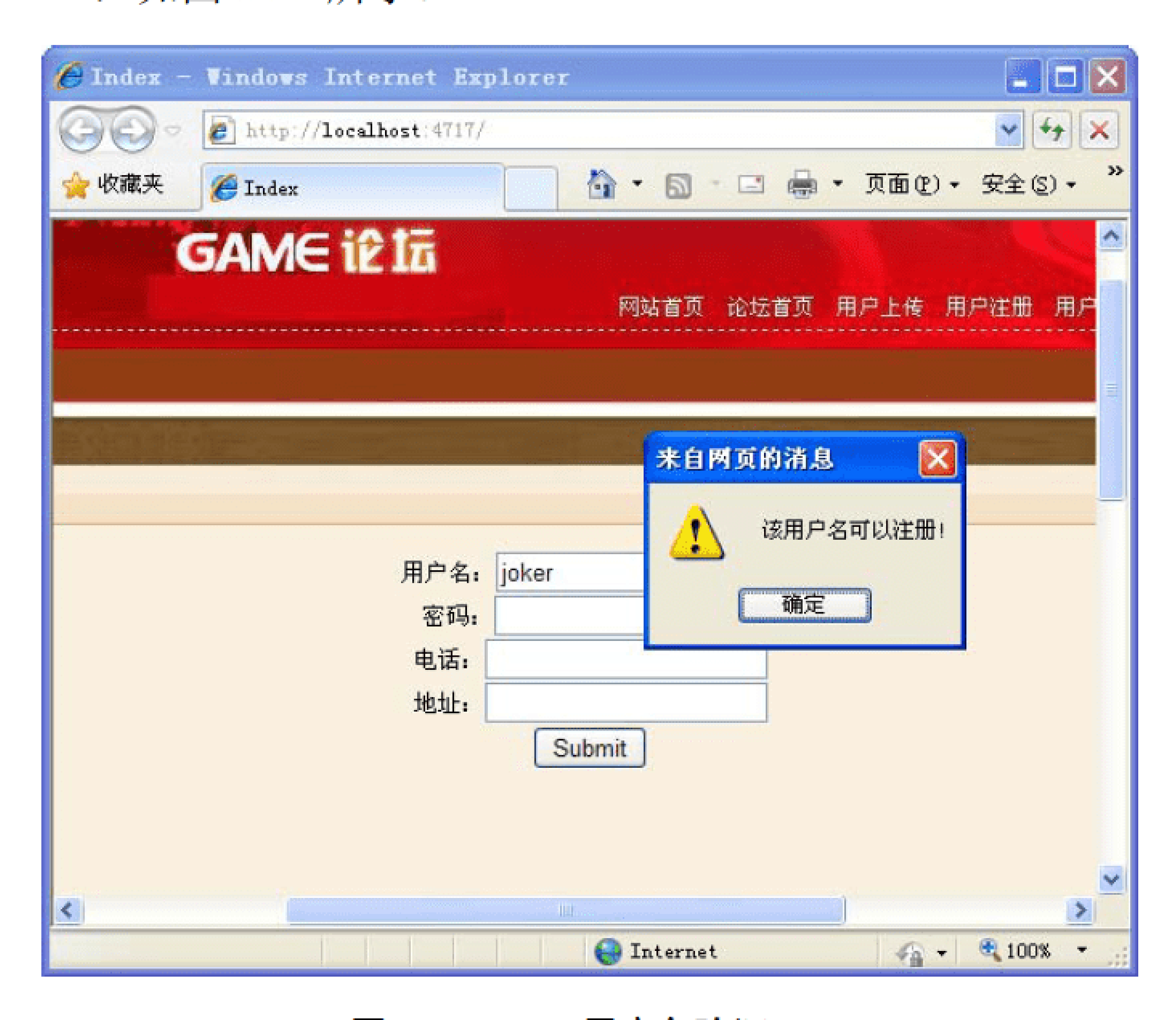


图 8-21　Ajax用户名验证(二)

8.7.5 实例分析



源码解析：

整个实例并不难，不过读者要学点 JQuery 的知识才能很好地理解那些代码。

通过这个实例我们发现，验证用户的 Action 可以直接在浏览器地址栏访问，这有点不爽。我们这样改一下即可防止用户这样访问。在 Action 代码上面跟加过滤器属性一样加一行代码[AcceptVerbs(HttpVerbs.Post)]，这样这个 Action 就只能以 Post 方式访问了，另外，我们的 JavaScript 代码做一下小的改动，把请求 Ajax 的 Get 方法改为 Post。再运行一下，效果一样，但是有效地防止了用浏览器直接访问的问题。

8.8 常见问题解答

8.8.1 ASP.NET MVC中能否使用WebForm中的服务器端控件



在 ASP.NET MVC 中还能使用 WebForm 中的服务器端控件吗？

网络课堂：http://bbs.itzcn.com/thread-9849-1-1.html

学了 MVC，看到视图又回归到那种“炸酱面”式的代码了，有点怀念 Web 服务器端的控件，比如 Repeater、DataList、DetailsView 等。

能不能在 View 中也使用这些服务器端控件呢？

【解决办法】：其实 ASP.NET MVC 和 ASP.NET WebForm 应用程序之间的区别就是视图的一些变化，ASP.NET WebForm 是将整个页面作为一个表单处理了。实际上 ASP.NET MVC 使用的也是 ASP.NET WebForm 的页面机制，所以在 ASP.NET MVC 中可以使用 WebForm 中的服务器端控件。

但是，因为在 MVC 中我们不习惯给控件添加事件(其实也不可以)，所以尽量不要使用太复杂的页面控件，省得出现一些莫名其妙的错误。

这里建议处理重复信息的时候使用迭代控件 Repeater。

8.8.2 关于ASP.NET MVC的初级问题



ASP.NET MVC 在 Controller 中的变量问题。

网络课堂：http://bbs.itzcn.com/thread-9859-1-1.html

我想请教一下 ASP.NET MVC 是用来做什么的？

主要还是用于大中型系统吗？

.NET 开发的初学者适合接触 ASP.NET MVC 吗？

【解决办法】：MVC 的开发效率很高，可以真正做到前后台分离。

MVC 的本质就是将业务逻辑层和业务视图分离，将数据层的数据通过逻辑层传给视图。

这样对于一般应用没有什么太大的优点，主要就是在企业级开发中，可以更好地做到任务分工，还有就是当系统庞大时更加易于维护。

如果是初学者，建议从 WinForm 学起，因为那里可以学到更多的基础知识。当你有了一定的基础，再来看 MVC 时，就会觉得这个框架很吸引人了，尤其是在 Ajax 方面。

8.9　习　　题

一、填空题

(1) 自定义的 Controller 需要继承自__________类。

(2) MVC 设计模式将应用程序按用户界面的功能划分为模型、视图、__________等三个模块。

(3) 默认情况下，控件代码:

```
Html.BeginForm(new { controller="News", action="List"})
```

将生成 HTML 文本:

```
<form action="__________" method="post">
```

(4) 重写 Controller 的__________方法可以对 Controller 进行异常捕获和处理。

(5) 在 MyProj.Filter 命名空间下声明一个过滤器 LogFilter，想把它用在 ProductController 里的 Edit 动作下面。补充以下代码完成操作:

```
[__________]
public ActionResult Success()
{
    return View();
}
```

二、选择题

(1) ASP.NET MVC 是一个__________。

A. 设计模式　　B. 框架　　C. 设计思想　　D. 类

(2) MVC 模式是一个处理__________层逻辑的设计模式。

A. 用户界面　　B. 业务逻辑　　C. 数据访问　　D. 数据模型

(3) 在 Action 里，使用下面的语句可以__________。

```
return View("Hello");
```

A. 将请求转发到名为 Hello 的 Controller 里的默认 Action

B. 重定向到名为 Hello 的 Controller

C. 重定向到当前 Controller 的名为 Hello 的 Action

D. 将返回与当前 Controller 同名的目录下的名为 Hello 的视图

(4) 下列说法正确的是__________。

A. ASP.NET 中 Model 必需放在 Models 目录下面

B. ASP.NET 中 Controller 必需放在 Controllers 目录下面

C. ASP.NET 中 View 必需放在 Views 目录下面

D. ASP.NET 中脚本文件必需放在 Scripts 目录下面

(5) 默认的 ASP.NET MVC 2 站点的访问路径是__________。

A. Default.aspx　　　　B. Home.aspx

C. Index　　　　D. /Home/Index

(6) 以下代码将在客户端代码中生成的结果为__________。

```
<%= Html.TextBox("name", "myName")%>
```

A. `<input id="name" name="myName" type="text"/>`

B. `<input id="name" name="name" type="text" value="myName"/>`

C. `<input id="myName" name="name" type="text"/>`

D. `<input id="myName" name="myName" type="text" value="name"/>`

三、上机练习

上机练习：GAME 论坛用户注册功能。

本练习使用 ASP.NET MVC 实现 GAME 论坛的用户注册功能。

注册功能需要用户提交用户名、密码、电话、地址等信息，提交后给用户提示注册结果。如图 8-22 和图 8-23 所示。



图 8-22　用户注册


Index - Windows Internet Explorer
http://localhost:4086/Home/Register
GAME 论坛
网站首页 论坛首页 用户上传 用户注册
用户名:
密码:
电话:
地址:
Submit
注册成功!

图 8-23　反馈信息



第 9 章　华丽的用户体验

内容摘要：

随着 Web 标准应用的日益广泛，现在越来越多网站开始注重用户体验，特别是 Web 2.0 网站，都通过为用户提供良好的视觉效果、交互体验来提高网站的粘性。

苹果、Google、百度、新浪、亚马逊……从本质上讲，它们的成功都是用户体验的成功。通过对用户体验的研究，这些企业成功地实现了对用户的吸引、粘连甚至是某种控制。

用户体验尽管起源于 IT 产品，但实际上任何个人、任何产品、任何行业的竞争背后都是用户体验的竞争。所以，掌握用户体验，就可以掌控未来大业。

在本章中，将以 ASP.NET 为主线，介绍 Flash 和 Flex 两种富表现层技术是如何提高用户体验的。例如，传递动态文本到 Flash 或获取 Flash 中的数据，制作 Flex 版的文件上传等。

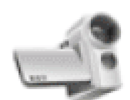
学习目标：

- 掌握 ASP.NET 数据传递到 Flash 的方法
- 掌握 ASP.NET 获取 Flash 数据的方法
- 掌握 Flash 如何加载外部数据
- 了解 Flex 与服务器端的 3 种交互方式
- 熟悉 HttpService 获取数据的方法
- 理解 XML 在数据交互时的作用
- 掌握 FileReference 类实现文件上传的方法

9.1 传递动态文本到Flash

大家都知道 Flash 中的文本框有三种类型——静态文本框、动态文本框和输入文本框。其中，静态文本框用于显示静态文本，其内容在 Flash 创建的时候已经固定；动态文本框用于显示动态文本，其内容可以通过 AS 实时改变；输入文本框用于输入文本，其内容为用户输入的任何文本或用户可以编辑的动态文本。

正是有了这几种类型的文本框，才使得 Flash 和用户的沟通交流变得更加容易，那么在 Flash 中动态文本框如何获取 ASP.NET 网页中的文本呢？这便是我们本节要解决的问题。

 **视频教学：** 光盘/videos/9/Flash_LoadVars.avi　　　　长度：13 分钟

9.1.1 基础知识——LoadVars类

LoadVars 类用于在 Flash 应用程序和服务器之间传输变量，它可以在下载时获取成功数据加载、进度指示和流数据的验证信息。

LoadVars 类主要提供了 3 个方法与服务器进行通讯，分别是 load()、send()和 sendAndLoad()。在使用的时候，必须先通过构造函数 new LoadVars()创建 LoadVaras 对象，然后才能调用 LoadVars 类的方法。语法格式如下：

```
my_lv:LoadVars = new LoadVars();
```

(1) load()方法

用于从指定的 URL 下载变量，分析变量数据，然后将结果变量放在 my_lv 中。其语法格式如下：

```
my_lv.load(url);
```

其中 url 参数即目标 URL。

(2) send()方法

用于将变量从 my_lv 对象发布到指定的 URL。其语法格式如下：

```
my_lv.send(url, targetObject[, method])
```

其中 url 参数即目标 URL；targetObject 参数即接收下载变量的 LoadVars 对象；method 参数即 HTTP 协议的 GET 或 POST 方法。

(3) sendAndLoad()方法

用于将 my_lv 对象中的变量发布到指定的 URL；并将下载服务器的响应，将其作为变量数据进行分析，然后将结果变量放在 my_lv 对象中。其语法格式如下(含义与 send 方法相同)：

```
my_lv.sendAndLoad(url, targetObject[, method])
```

LoadVars 类还有一个 onLoadd()事件处理函数，当 load()或 sendAndLoad()操作结束时调用。如果该操作成功，my_lv 将填充为该操作所下载的变量，而这些变量将在调用此处理函数时变

为可用。其语法格式如下：

```
my_lv.onLoad = function(success) { //此处是您的语句 }
```

其中 success 参数指示加载操作是成功完成(true)还是以失败结束(false)。

9.1.2　实例描述

如果你经常上网，就可以发现如今越来越多的个人网站变成了 Flash 全站。它们表现得很酷，很有互动性，或者很能营造氛围，看得自己真是口水直流啊！

那么什么是 Flash 全站呢，Flash 全站就是将网站“Flash 化”，把所有需要用到的数据、文档、资料、图片都用 Flash 制作、输出成 SWF 文件，然后嵌入到 HTML 文件中。用户浏览时就不需要一个页面一个页面地打开，只要打开一个主界面，就可以浏览全站的内容。

不过一个好的网站光靠华丽的外表是没用的，关键是如何很快地更新网站的内容，如何很好地与用户产生互动。那么，一个 Flash 网站是如何更新网站内容的呢？利用 ASP.NET 能否实现为 Flash 动态添加内容呢？下面，就让我们通过实例，来解答一下读者心中的疑问。

9.1.3　实例应用

【例 9-1】传递动态文本到 Flash。

打开 Flash CS4，在 Flash 文件中添加一个图层，并命名为“addtxt”。

在 addtxt 图层中添加两个动态文本框，并分别设置实例名为 title 和 content，再设置每个实例的变量为 dtitle 和 dcontent。

接下来添加一个名为“action”的图层，并打开该层的“动作”窗口。

在这里编写如下的 ActionScript 代码：

```
var mainadd = new LoadVars();            //定义读取函数
mainadd.load("addContent.aspx");         //指定动态数据来源

mainadd.onLoad = function(success)       //从 aspx 文件中读取动态数据，并判断是否成功
{
    if(success)
    {
        dtitle=mainadd.addtitle; //把 mainadd 获取到的变量 addtitle 赋给变量 dtitle
        dcontent =
          mainadd.addcontent;//把 mainadd 获取到的变量 addcontent 赋给变量 dcontent
    }
    else
    {
        gotoAndPlay("fail");
    }
}
```

至此，Flash 部分设计完成。

下面打开 VS2008 新建一个"addContent.aspx"网页用来生成动态数据。删除网页前台除第一行的全部代码，然后在代码页的 Page_Load()中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    string Title = "FLEX 从入门到精通";
    string Content = "窗内网 FLEX+ASP.NET 视频教程系统讲解了 FLEX+ASP.NET 基础知识，
并通过大量的案例引导学生循序渐进地深入掌握 FLEX+ASP.NET 开发实际项目，为方便学生动手操作，
每个视频界面提供案例下载，学生在观看视频的同时，如果对该案例有兴趣，可以下载到本机进一步分
析学习！";
    Response.Write(
      String.Format("addtitle={0}&addcontent={1}", Title, Content));
}
```

这里定义了两个变量，然后直接赋值给它们。再复杂一点，我们也可以通过数据库给它们赋值，这样的话，Flash 便可以通过 ASP.NET 间接地与数据库交互了。

最后，我们新建一个"Flashtest.aspx"网页，在网页中插入 Flash.swf，完成实例的制作，具体代码参考本节的源文件。

9.1.4 运行结果

经过几番编辑工具的转换，现在有些迫不及待地想看看劳动成果了。

具体效果如图 9-1 所示。好了，为了比较一下效果，大家可以直接在 Flash Player 中打开 Flash.swf 文件进行对比。



图 9-1 将动态文本传到Flash的页面效果

9.1.5　实例分析



源码解析：

从效果可以看出，Flash 中确实动态显示了 aspx 页面输出的内容，那么 aspx 是怎样把数据传给 Flash 的呢？代码中好像看不出来 Flash 和 aspx 页面产生交互啊。

读者注意了，我们运行一下 addContent.aspx 页面，网页上输出“addtitle=FLEX 从入门……”，对其中的“addtitle”，是不是似曾相识呢？

其实我们在编写 Flash 的代码时就用到了“mainadd.addtitle”。哦，原来在这里 Flash 是从 aspx 的输出内容中通过变量获取动态内容的！在这里还要提醒一下，如果有多个变量，变量之间要通过“&”符号隔开。

9.2　获取Flash中的动态文本

在 ActionScript 3.0 中，LoadVars 类功能替换为 URLLoader、URLRequest、URLStream 和 URLVariables 类。每个类的主要功能如下。

- URLLoader：以文本、二进制数据或 URL 编码变量的形式从 URL 下载数据。
- URLRequest：可捕获单个 HTTP 请求中的所有信息。
- URLVariables：可以在 Flash 应用程序和服务器之间传输变量。
- URLStream：提供对下载 URL 的低级访问。



视频教学：光盘/videos/9/navigateToURLMethod.avi　　长度：10 分钟

9.2.1　基础知识——navigateToURL()方法

navigateToURL()为 ActionScript 3.0 中的公共函数。其主要功能是在包含 Flash Player 容器的应用程序(通常是一个浏览器)中，打开或替换一个窗口。该方法可以从一个指定的 URL 加载一个文件到浏览器窗口，也可以用来在 Flash 和 JavaScript 之间建立通信。navigateToURL()方法位于 Flash.net 包中，其语法格式如下：

```
navigateToURL(request:URLRequest, window:String):void
```

其中，request 参数是一个 URLRequest 对象定义的目标；window 参数定义内容加载到浏览器后浏览器窗口打开的方式。如果没有为此参数指定值，浏览器将打开一个新的窗口。



window 参数的值与 HTML 中 target 的值相同，可选值有 self、_blank、_parent 和 _top。

示例：在一个新的浏览器窗口中打开 http://www.adobe.com，并将在 URLVariables 对象中所捕获的用户会话数据传递给 Web 服务器。代码如下：

```
var url:String = "http://www.adobe.com";
var request:URLRequest = new URLRequest(url);
var variables:URLVariables = new URLVariables();
variables.exampleSessionId = new Date().getTime();
variables.exampleUserLabel = "Your Name";
request.data = variables;
navigateToURL(request, -blank);
```

这个方法也可以用来执行 JavaScript，例如：

```
var url:URLRequest = new URLRequest("javascript:window.close()");
navigateToURL(url, _blank);
```

9.2.2 实例描述

在上一节中，我们讲到了传递动态数据到 Flash，而在实际运用中，Flash 只有这一个功能是远远不够的，因为这还不能算是真正意义上的动态。能够很好地同用户“交互”，才是一个网站永恒不变的话题。“交互”应该是一个“礼尚往来”的概念，也就是说，既“有来”，也得“有往”。

其实上一节中，我们讲到的“传递”便是“有往”，那么这一节中就说说如何“获取”，即“有来”。

9.2.3 实例应用

【例 9-2】获取 Flash 中的动态文本。

首先，在 Flash CS4 中新建一个名为“addMusic”的图层。

在该图层上，我们添加静态文本框、文本框、按钮以及一个 label 标签，并设置相关属性值，最终的布局效果如图 9-2 所示。另外，为了起到美化效果，还可以添加其他层。



图 9-2　addMusic图层布局

接下来我们添加一个动作图层，命名为“as”。在该图层的“动作”窗口中编写以下代码：

```
System.useCodePage = true;
stop();
//定义事件处理函数
addBtn.addEventListener(MouseEvent.CLICK, clickHandler);

function clickHandler(event:MouseEvent):void
{
     trace("clicked (localX:" + event.localX + ", localY:" + event.localY + ")");
     var url:String = "Default.aspx";                //指定接收动态文本的URL
     var request:URLRequest = new URLRequest(url); //创建URLRequest类
     var variables:URLVariables = new URLVariables();  //创建URLVariables类
     variables.cId = lblId.text;
     variables.cName = txtName.text;
     variables.cSinger = txtSinger.text;
     variables.cAlbum = txtAlbum.text;
     variables.cYear = txtYear.text;
     variables.cWord = txtWord.text;
     variables.cMusic = txtMusic.text;
     request.data = variables;
     request.method = "post";                    //设定传送方式
     navigateToURL(request, "_self");            //转发请求到指定URL
};
```

Flash 动画设计完毕。接下来我们新建一个“default.aspx”网页。在该页面的适当位置插入刚设计好的 Flash 文件，然后在该页面中添加一个表格，用于显示从 Flash 获取到的动画文本。该段代码如下：

```
<table class="listing" cellpadding="0" cellspacing="0">
   <tr>
      <th class="first" width="60">歌曲编号</th>
      <th>歌曲名称</th>
      <th>演唱歌手</th>
      <th>出自专辑</th>
      <th>发行年份</th>
      <th>作词人</th>
      <th class="last">作曲人</th>
   </tr>
   <tr>
      <td class="first style1">
      <asp:Literal ID="ltID" runat="server"></asp:Literal></td>
      <td><asp:Literal ID="ltName" runat="server"></asp:Literal></td>
      <td><asp:Literal ID="ltSinger" runat="server"></asp:Literal></td>
      <td><asp:Literal ID="ltAlbum" runat="server"></asp:Literal></td>
      <td><asp:Literal ID="ltYear" runat="server"></asp:Literal></td>
      <td><asp:Literal ID="ltWord" runat="server"></asp:Literal></td>
      <td class="last">
      <asp:Literal ID="ltMusic" runat="server"></asp:Literal></td>
```

```
    </tr>
</table>
```

为了接收从 Flash 中传递过来的动态文本，我们需要在该页面的 Page_Load()事件中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    ltID.Text = Request["cId"];                //显示歌曲编号
    ltName.Text = Request["cName"];            //显示歌曲名称
    ltSinger.Text = Request["cSinger"];        //显示演唱歌手
    ltAlbum.Text = Request["cAlbum"];          //显示出自专辑
    ltYear.Text = Request["cYear"];            //显示发行年份
    ltWord.Text = Request["cWord"];            //显示作词
    ltMusic.Text = Request["cMusic"];          //显示作曲
}
```



如果在运行的过程中出现中文乱码，可以在 Web.config 文件中将编码格式改为 gb2312。

9.2.4 运行结果

忙活了半天，读者想必都在期盼着看到结果的这一刻吧！我们赶快保存一下文件来运行网页吧，效果如图 9-3 所示。嗯，猛一看，页面中的 Flash 与传统中的网页表单没有什么区别啊，但是我们通过鼠标右键单击发现，这确确实实是一个 Flash 文件。那好，我们来输入点内容，看看效果如何。



图 9-3　我们向Flash中添加内容

输入完内容后，单击“添加新歌曲”按钮，可以看到页面的表格中显示出了我们刚添加的内容，如图 9-4 所示。



图 9-4　单击“添加新歌曲”按钮后的页面效果

9.2.5　实例分析



源码解析：

通过本实例，我们弄清楚了 Flash 向网页中传送数据的过程。这个过程有些复杂，但并不难。主要用到 URLRequest 和 URLVariables 两个对象，以及 navigateToURL()方法。它们的工作流程如下。

首先创建名为 request 的 URLRequest 实例，同时将应用程序的 URL 作为参数。然后创建一个名为 variables 的 URLVariables 实例，并对它的属性进行赋值。接下来将 variables 实例赋给了 request 实例的 data 属性。然后设定传送方式后调用 navigateToURL，在数据传送到指定 URL 后在原来窗口中打开。

9.3　实现动态下拉菜单

随着网站内容的日益丰富，一个简单的导航栏已不能满足网站内容分类导航的需求。于是，越来越多的网站出现了下拉菜单。关于如何实现下拉菜单，我们在 Baidu 或 Google 上一搜就会知道，有很多技术都能实现。其实“条条大路通罗马”，只要我们的功能实现了，导航方便了，界面美观了，不管什么技术都是好技术。

在众多的技术当中，要想实现动态，XML 便是个不能不提的功臣。那么在 Flash 中，XML 类起着什么样的作用？它是否能与一个 ASP.NET 输出的 XML 相结合呢？

学习完本节的内容，这一系列的问题都会最终找到答案。



视频教学：光盘/videos/9/Flash_XML.avi　　　　长度：10 分钟

9.3.1 基础知识——XML类

使用 XML 类可以加载、分析、发送、生成和操作 XML 文档树。可以通过 new 关键字调用构造函数来创建一个 XML 类实例，语法格式如下：

```
var my_xml:XML = new XML();
```

XML 类与 LoadVars 类相似，也有 load()、send()和 sendAndLoad()方法，以及 onLoad()事件。但是 LoadVars 类的方法有返回值，而 XML 类的方法无返回值。

在 XML 类的属性中，最常用的有 3 个，分别是 firstChild、childNodes 和 attributes。下面对它们做简单的介绍。

- firstChild：该属性用于计算指定的 XML 对象，并引用父级节点的子级列表中的第一个子级。
- childNodes：用于指定 XML 对象的子级组成的数组。数组中的每个元素都是对表示子级节点的 XML 对象的引用。这是一个只读属性，无法用于操作子级节点。
- attributes：一个包含指定 XML 对象的所有属性的关联数组。

例如，下面的代码演示了 XML 属性的用法：

```
str = "<mytag name=\"Val\"> intem </mytag>";
doc = new XML(str);
y = doc.firstChild.attributes.name;
trace(y);                                    //输出 Val
doc.firstChild.attributes.order = "first";
z = doc.firstChild.attributes.order;    //输出 first
trace(z);
```

9.3.2 实例描述

在前面的章节中，我们提到过 Flash 全站的一个基本概念，大家都知道华丽的网站固然能吸引网民的眼球，但如果没有丰富的内容作基础，那么再“酷”的网站最后也不过是落得个“华而不实”的称谓罢了。但是，如果一个网页内容太过丰富，涉及的主题又比较多，问一下读者：这样“一锅煮”的网页你愿意浏览吗？

要解决这个问题，其实很简单，给网站加一个漂亮的导航，不同主题内容放到一个网页中。那么在 Flash 做的网页中，这个导航是如何实现的呢？

下面带着这个问题，进入本节的实例——动态实现 Flash 下拉导航菜单。

这里再说点题外话，本实例与真正意义上的全 Flash 网站导航还有一些区别，后者在导航的时候是导航到 Flash 的另一个场景中去了，而不是一个网页，不过二者大同小异。

9.3.3 实例应用

【例 9-3】动态实现 Flash 下拉菜单。

打开 VS2008，新建一个“xmlMenu.aspx”网页。它将作为下拉菜单的数据源，输出一个 XML 类型的内容。在后台文件中添加如下代码：

```
string AddNodeBookType(string nodeName, string nodeList) //添加 BookType 节点
{
    return string
      .Format("<booktype name=\"{0}\">{1}</booktype>", nodeName, nodeList);
}
string AddNodeBookName(string bookName, string bookUrl) //添加 BookName 节点
{
    return string
      .Format("<bookname subname=\"{0}\" url=\"{1}\"/>", bookName, bookUrl);
}
```

上代码创建了两个方法，分别用于创建 BookType 节点和 BookName 节点，其中 BookName 节点被包含在 BookType 节点中。

我们知道，一个标准 XML 有且只能有一个根节点。那么，接下来就定义下拉菜单 XML 的根节点以及其中保存的数据。在 Page_Load()中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    Response.ContentType = "text/xml";
    string childNode, xmlNode;
    childNode = AddNodeBookName("C#从入门到精通", "#");
    childNode += AddNodeBookName("FLEX 从入门到精通", "#");
    childNode += AddNodeBookName("J2EE 完全自学手册", "#");
    xmlNode = AddNodeBookType("软件开发", childNode);
    childNode = AddNodeBookName("3ds Max 技法精粹", "#");
    childNode += AddNodeBookName("Maya 无师自通", "#");
    childNode += AddNodeBookName("ZBrush 无师自通", "#");
    xmlNode += AddNodeBookType("三维动画", childNode);
    childNode = AddNodeBookName("PHP 网络大讲堂", "#");
    childNode += AddNodeBookName("ASP.NET 从入门到精通", "#");
    xmlNode += AddNodeBookType("WEB 编程", childNode);
    childNode = AddNodeBookName("SQL 2008 完全学习手册", "#");
    childNode += AddNodeBookName("Oracle 11g 完全学习手册", "#");
    xmlNode += AddNodeBookType("数据库", childNode);
    childNode = AddNodeBookName("Windows XP 视频教程", "#");
    childNode += AddNodeBookName("Word 2007 视频教程", "#");
    xmlNode += AddNodeBookType("基础应用", childNode);
    string xml = string.Format("<root>{0}</root>", xmlNode);
    Response.Write(xml);
}
```

添加完成后，我们先运行一下，看看在页面中输出的是不是一个标准的 XML 文件。剩下的工作在 Flash CS4 中完成。

新建两个“影片剪辑”元件，分别命名为“menu”和“submenu”，并分别设其标识符为“mc”和“xialamc”。

下面依次在每个元件中各添加一个动态文本框，并分别设置其变量为“txt”和“subtxt”，分别用于显示菜单以及子菜单。

接下来，再新建一个动作图层，将其命名为“as”，在其“动作”窗口中添加如下代码：

```
fscommand("allowscale", "false");        //固定大小，禁止缩放
var my xml:XML = new XML();          //创建一个新的空 XML 对象
my_xml.ignoreWhite = true;            //取消空白节点输出
my_xml.load("/xmlMenu.aspx");         //加载 xml，aspx 页面会输出一个 xml 类型的文件
my_xml.onLoad = function(ok:Boolean)   //判断是否加载成功，如果成功，则继续前进
{
    if (ok) {
        xml_Length = my_xml.firstChild.childNodes.length; //读取 xml 子节点的长度
        for (i=0; i<xml_Length; i++) {
             root.attachMovie("mc", "mc"+i, i);  //调用菜单元件"mc"
            _root["mc"+i]._x = i*80+120;         //定位菜单
            _root["mc"+i]._y = 100;
            _root["mc"+i].txt =
              my_xml.firstChild.childNodes[i].attributes.name; //读取菜单名称
        }
    }                                   //根据 xml 子节点来设置菜单
    else {
        trace("加载 XML 失败!");
    }
};
```

在导航菜单栏中，当以鼠标单击菜单的时候，我们让 Flash 弹出一个子菜单，当鼠标离去的时候删除子菜单。在子菜单中，当鼠标点击的时候，导航到对应的网址。下面给出实现该功能的 AS 代码：

```
//当鼠标单击菜单时，显示该菜单的子菜单
_root.onMouseDown = function()
{
    for (i=0; i<xml_Length; i++)
    {
        xml_Length2 = my_xml.firstChild.childNodes[i].childNodes.length;
        if (_root["mc"+i].hitTest(_root._xmouse, _root._ymouse))
        {
            if (my_xml.firstChild.childNodes[i].hasChildNodes())
            {
                for (j=0; j<xml Length2; j++)
                {
                    //调用子菜单元件 xialamc
                    _root.attachMovie("xialamc", "xialamc"+i+j, i+j+50);
                    _root["xialamc"+i+j]._x = i*80+165; //设置子菜单的坐标
                    _root["xialamc"+i+j]._y = j*30+125;
                    //读取子菜单名称
                    _root["xialamc"+i+j].subtxt = my_xml.firstChild
                      .childNodes[i].childNodes[j].attributes.subname;
                }
```

```
            }
        }
        else { //否则删除该菜单下的子菜单
            for (j=0; j<10; j++)
            {
                removeMovieClip(_root["xialamc"+i+j]);
            }
        }
    }
    if(k){      //当 k 为真时，导航到对应网址
        getURL(my_url1,_blank);
    }
};
```

最后，我们再来美化一下导航菜单，当鼠标经过的时候，我们改变一下它的背景颜色。该段 AS 代码如下：

```
_root.onMouseMove = function()
{
    k = false;
    //当鼠标触及菜单时，菜单改变颜色
    for (i=0; i<xml_Length; i++) {
        if (_root["mc"+i].hitTest(_root._xmouse, _root._ymouse))
        {
            _root["mc"+i].gotoAndStop(2);
        }
        else {   //否则仍旧为默认颜色
            _root["mc"+i].gotoAndStop(1);
        }
        //鼠标触发子菜单时，改变子菜单的颜色
        for (j=0; j<10; j++)
        {
            if (_root["xialamc"+i+j].hitTest(_root._xmouse, _root._ymouse))
            {
                 root["xialamc"+i+j].gotoAndStop(11);
                //读取 url 值，并赋值给变量 my_url1
                my_url1 = my_xml.firstChild
                  .childNodes[i].childNodes[j].attributes.url;
                k = true; //设置 k 为真
            }
            else if(_root["xialamc"+i+j]._currentframe!=10
              && _root["xialamc"+i+j]._currentframe!=11)
            {
                _root["xialamc"+i+j].play();
            }
            else { //将 xialamc 内时间轴停在第 10 帧
                _root["xialamc"+i+j].gotoAndStop(10);
            }
        }
```

```
    }
}
```

注意 读者注意一下上段代码中 k 的作用，它是为上面鼠标 MouseDown 事件服务的。目的是保证只有鼠标在子菜单上单击的时候，才导航到对应网址。

至此，Flash 部分也设计完成了。结束时我们再新建一个“FlashMenu.aspx”网页，并在网页中插入设计好的 Flash 文件。

9.3.4 运行结果

完成这个简单的 Flash 动态下拉菜单导航栏的制作后，快来看看效果如何。如图 9-5 所示为本实例运行的效果。



图 9-5 Flash动态下拉菜单的效果

9.3.5 实例分析



源码解析：

纵观本实例，有两个地方需要大家理解。

其一是，如何用 ASP.NET 在页面输出一个 XML 类型的文件？要解决这个问题，我们采取由局部到整体的思想，我们首先添加子节点的列表，然后再在子节点外面套上父节点，最后在父节点外面套上根节点并输出。

其二是，Flash 如何动态地从 aspx 页面获取数据？其实这个问题的基本思路我们已经在前面做过讲解，本节需要多注意的是 Flash 文件的设计细节问题，比如说在什么位置显示菜单、显示菜单时调用哪个元件等。

9.4　实现Flex通信录

通过在 Flex 中使用组件或者 ActionScript 与一种服务器端相结合，将使网页显示更加精彩和丰富。本节以 ASP.NET 为例，讨论两者是如何结合的，并实现通讯录的功能。

 视频教学：光盘/videos/9/Flex_httpService.avi　　　长度：12 分钟

9.4.1　基础知识——Flex与外部数据的交互方式

数据交互是指 Flex 与其他服务器端程序进行数据交换，包括传递数据给其他程序和接收其他程序返回的数据。

Flex 的数据交互方式主要有 3 种，分别是 HttpService、WebService 和 RemoteObject。表 9-1 中分别描述了这 3 种交互方式的特性。

表 9-1　交互方式

通信方式	通信协议	交互数据格式
HttpService	常用的 HTTP 协议	XML
WebService	SOAP 协议	XML
RemoteObject	Flex 自定义的高效二进制数据通讯协议：AMF	任意(可以是数字、字符串或者图片等)

同时，这 3 种方式也同时也存在一些优缺点，表 9-2 中对它们进行了比较。

表 9-2　交互方式比较

通信方式	优　点	缺　点
HttpService	数据格式通用，便于不同应用系统间交换数据	① 数据在发送前需转换成 XML 格式，接收后要解析 XML 数据。哪怕是只发送一个简单的数字也要如此。② 在处理复杂数据类型(如图片和对象)的时候，非常不方便
WebService	同上	同上
RemoteObject	能够处理各种类型的数据，速度快	需要专门的服务器端软件 LCDS(FDS)或 AmfPHP

9.4.2　实例描述

通讯录通常是用来保存、管理和查询联系人信息的一个简单信息系统，通过它用户能够快速、有效地获取联系人的资料。如家庭电话、客户的办公电话以及移动电话、同学的 E-mail 和学校网址等。

在本节中，将选择 HTTPService 方式与 ASP.NET 和数据库进行交互，最终实现一个简单

的通讯录应用实例。

整体制作过程为：设计数据库→编写 ASP.NET 服务器程序→编写 Flex 客户端程序。

9.4.3 实例应用

【例 9-4】实现 Flex 通讯录。

(1) 在创建项目之前，首先考虑如何保存通讯录的数据。在本示例中，使用 Access 数据库来存放数据，数据库名称为“Contacts”，数据表为“persons”，最终设计如图 9-6 所示。

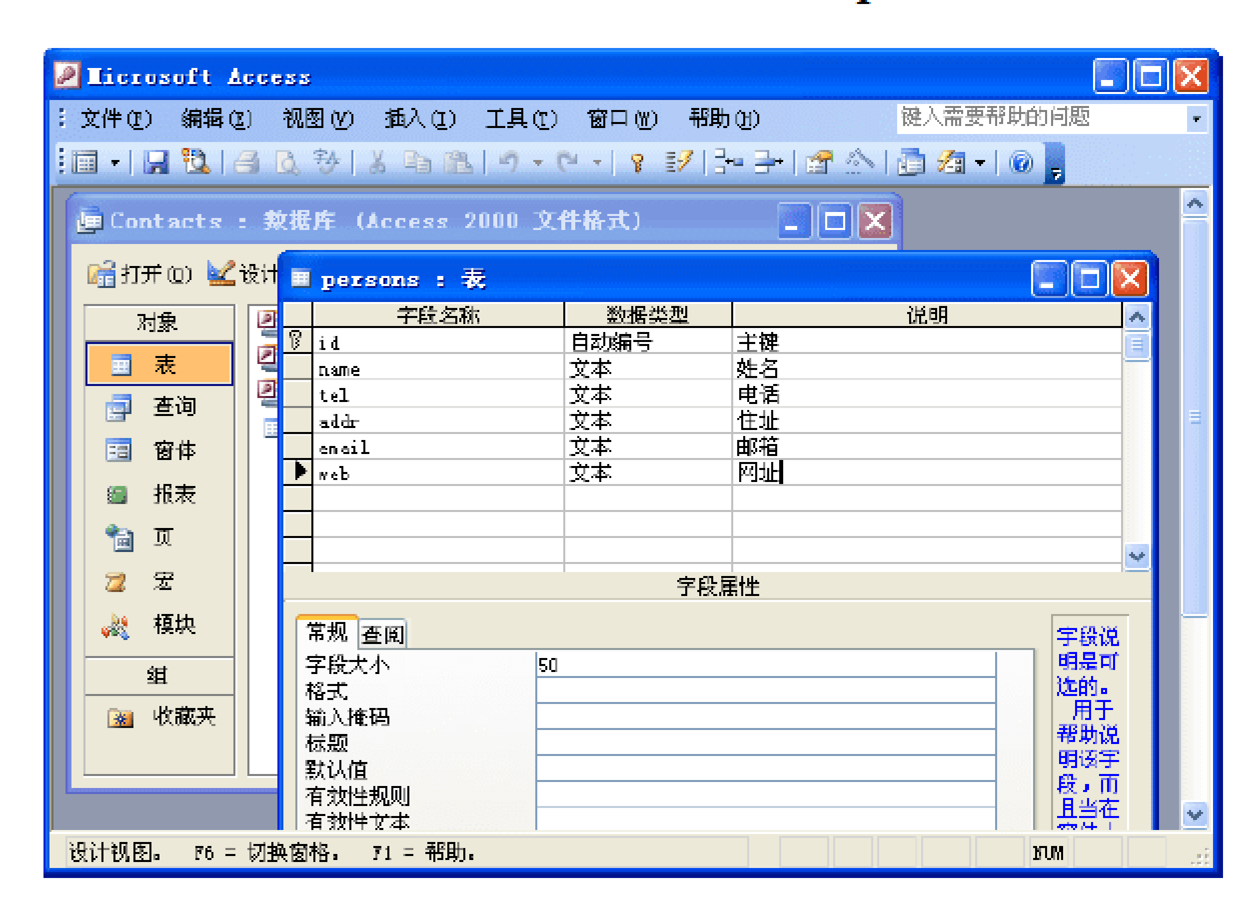


图 9-6 设计数据库和表

(2) 数据表创建完成后，顺便添加一些测试数据。下面创建通讯录的服务器端 ASP.NET 程序。新建名为“Contacts”的 ASP.NET 网站，再将上一步创建的 Access 数据库放在网站的根目录下。

(3) 我们知道，Flex 不能直接获取 ASP.NET 中的数据，必须借助于一种第三方存储格式，那就是 XML。在 Default.aspx.cs 中创建一个 getAllDataToXml()方法返回 XML 数据，这些数据包括所有的联系人信息：

```
private void getAllDataToXml()          //将所有记录转换成 XML 并输出
{
   string strXml = "<AllPerson>";
   string ConnString = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source="
     + Server.MapPath("~/Contacts.mdb");
   OleDbConnection con = new OleDbConnection(ConnString);
   con.Open();
   OleDbCommand cmd = new OleDbCommand("select * from persons", con);
   OleDbDataReader sdr = cmd.ExecuteReader();
   while (sdr.Read())
   {
      strXml += "<Person>"
        + "<id>" + sdr[0].ToString() + "</id>"
        + "<name>" + sdr[1].ToString() + "</name>"
        + "<tel>" + sdr[2].ToString() + "</tel>"
```

```
            + "<addr>" + sdr[3].ToString() + "</addr>"
            + "<email>" + sdr[4].ToString() + "</email>"
            + "<web>" + sdr[5].ToString() + "</web>"
            + "</Person>";
    }
    strXml += "</AllPerson>";
    sdr.Close();
    con.Close();
    Response.Write(strXml);
}
```

调用该方法后，运行效果如图 9-7 所示，说明返回的 XML 格式良好。

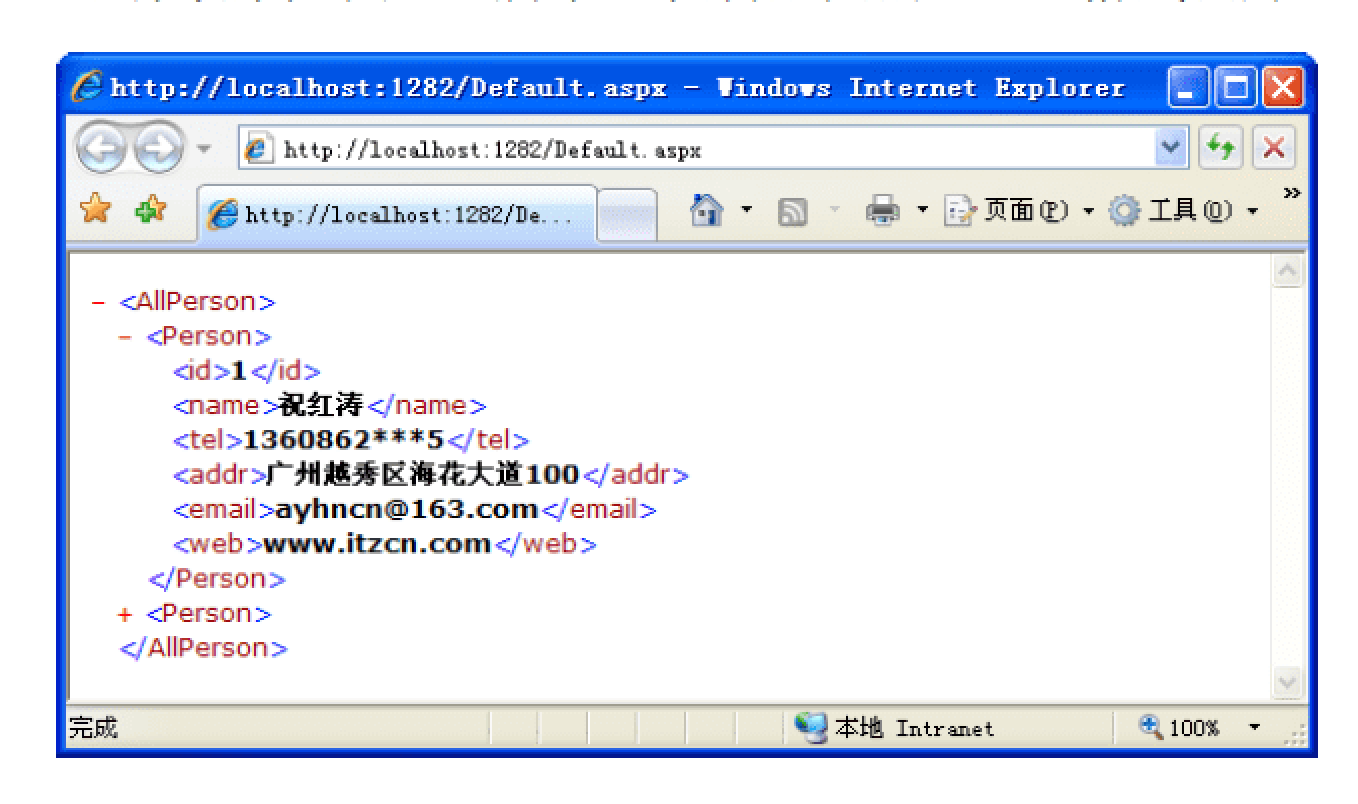


图 9-7　测试getAllDataToXml()方法

(4) 在前面已经完成了数据库的创建、项目的创建以及服务器脚本的编写。

那么接下来的工作都在 Flex 中进行，首先是制作出通讯录的界面，接下来再编写实现代码。Flex 中的第一步是创建名为“Contacts”的项目。

(5) 一个优秀的页面布局是网站成功的一半。在本实例中，首先修改 Application 组件的 verticalAlign 属性为 top、horizontalAlign 属性为 center、layout 属性为 horizontal，使容器水平居中排列，垂直顶部排列。

(6) 然后添加两个 Panel 组件。第 1 个 Panel 组件用于显示通讯录中的联系人列表，组件的布局代码如下：

```
<mx:Panel layout="vertical" title="我的所有联系人" verticalAlign="top"
  horizontalAlign="center" width="30%" height="100%">
    <mx:List id="EmpList" width="100%" labelField="name"
      itemClick="itemClick_handle(event)"  height="100%" >
    </mx:List>
</mx:Panel>
```

如上述代码所示，在 EmpList 组件中单击时会调用 itemClick_handle(event)函数，该函数将获取完整的信息然后在右侧的 Form 中进行显示。

(7) 第 2 个 Panel 组件用于显示详细的通讯信息，最终组件的布局如下面的代码所示：

```
<mx:Panel width="70%" height="100%" layout="vertical"  title="详细信息"
  cornerRadius="6" barColor="#521174" horizontalAlign="center"
```

```
 verticalAlign="top">
   <mx:Form>
      <mx:FormItem label="编    号: ">
         <mx:Label id="lbl_id" width="90"/>
      </mx:FormItem>
      <mx:FormItem label="姓    名: ">
         <mx:TextInput id="txt_name" width="220"/>
      </mx:FormItem>
      <mx:FormItem label="电    话: ">
         <mx:TextInput id="txt_tel" width="220"/>
      </mx:FormItem>
      <mx:FormItem label="住    址: ">
         <mx:TextInput id="txt_addr" width="220"/>
      </mx:FormItem>
      <mx:FormItem label="邮    箱: ">
         <mx:TextInput id="txt_email" width="220"/>
      </mx:FormItem>
      <mx:FormItem label="网    址: ">
         <mx:TextInput id="txt_web" width="220"/>
      </mx:FormItem>
      <mx:HBox horizontalGap="5">
         <mx:Button label="添 加" id="btnadd"/>
         <mx:Button label="更 新" id="btnUpdate"/>
         <mx:Button label="删 除" id="btndel" />
      </mx:HBox>
   </mx:Form>
</mx:Panel>
```

上述代码中，使用 Form 组件来将通讯录的详细信息都展现出来，同时还包含 3 个按钮，用于执行管理操作。

如图 9-8 所示为包含两个 Panel 组件时的布局。

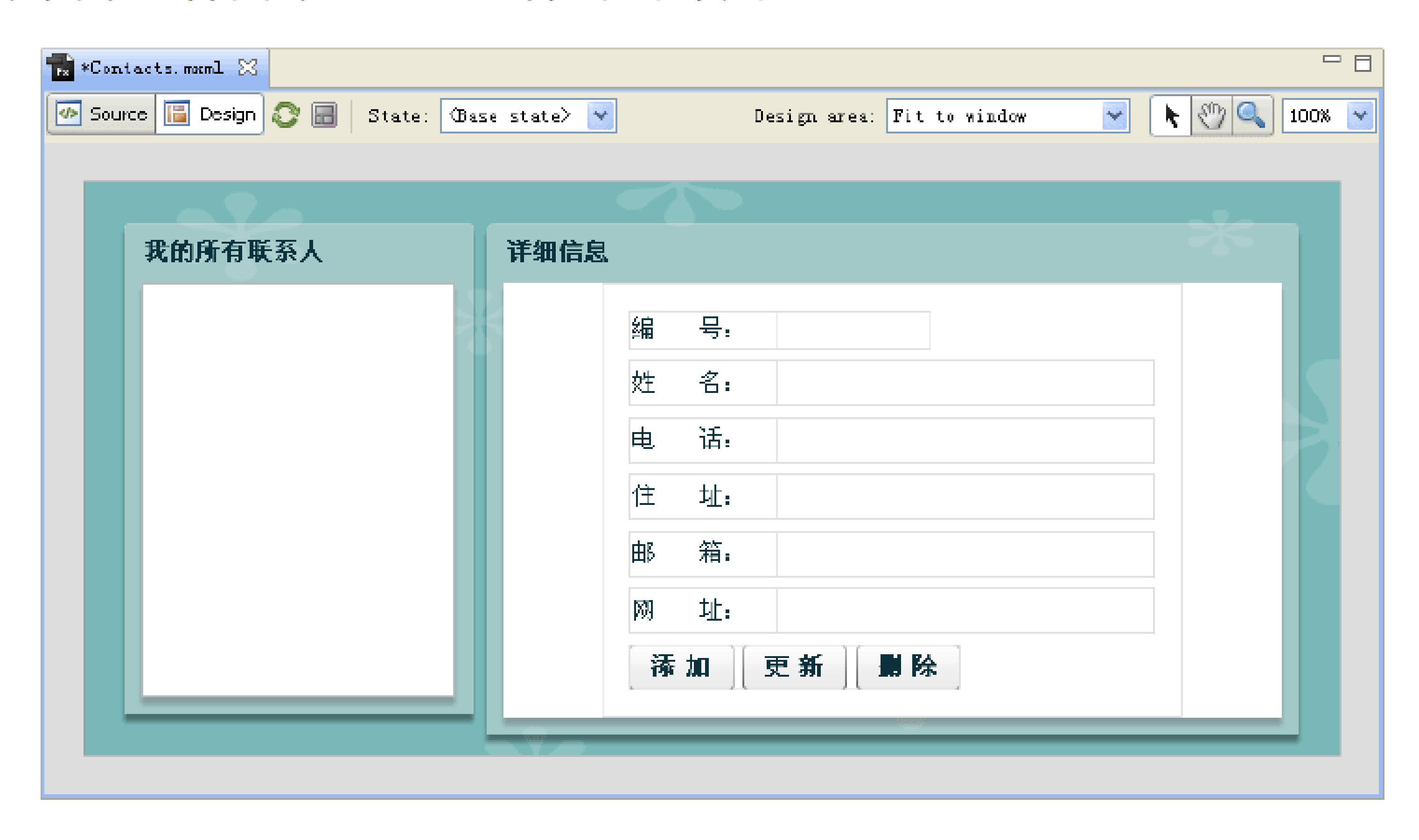


图 9-8　设计Flex客户端布局

(8)　为 Application 组件添加 creationComplete 属性，指定页面加载完成后执行 initApp() 函数。

(9)　使用 ActionScript 创建 initApp()函数，然后编写显示通讯列表的代码，最终如下所示：

```
<mx:Script>
    <![CDATA[
        import mx.events.ListEvent;                 //添加引用
        import mx.rpc.events.ResultEvent;
        import mx.rpc.events.FaultEvent;
        import mx.controls.Alert;
        import mx.rpc.http.*;

        var myService:HTTPService = new HTTPService();
        private function initApp():void {          //声明函数
            btnUpdate.enabled = false;              //禁用“更新”按钮
            myService.url = "Default.aspx";
            myService.method = "Get"
            myService.resultFormat = "e4x";
            myService.addEventListener(ResultEvent.RESULT,ResultHandle);
            myService.addEventListener(FaultEvent.FAULT,faultHandle);
            myService.send();           //发送 HttpService
        }
    ]]>
</mx:Script>
```

上述代码中指定以 GET 方式获取 Default.aspx 的返回结果，并指定了结果和错误事件的监听函数。

(10) 编写结果处理函数 ResultHandle()。该函数将从 ASP.NET 中获取的文本转换成 XML，并将其中的 Person 节点列表绑定到 id 为 EmpList 的 List 组件，作为 List 组件的数据源。由于 List 组件的 labelField 属性设置为“name”，因此将显示所有名片列表的名称。这部分代码如下所示：

```
//结果处理函数
private function ResultHandle(e:ResultEvent):void {
    trace(e.result.toString());
    var returnValue:XMLList = XML(e.result).Person;     //获取返回的结果
    EmpList.dataProvider = returnValue;                 //绑定到 EmpList 组件
    myService.disconnect();
}
```

(11) 错误处理函数用于实现当获取数据结果出错时，弹出一个对话框显示错误的具体信息，具体代码如下：

```
//错误处理函数
private function faultHandle(e:FaultEvent):void {
    var err:String = "发生错误";
    mx.controls.Alert.show(e.message.toString(), err); //弹出错误信息
}
```

(12) 在 EmpList 组件上有一个 itemClick 事件。该事件触发时，将被选择的联系人详细信息转换成一个 XML 文档，然后再将所有信息赋值到相应的文本框，代码如下：

```
private function itemClick_handle(evt:ListEvent):void {
    btnUpdate.enabled = true;    //使“更新”按钮可用

    trace(evt.target.toString());
    trace(EmpList.selectedItem.toString());

    //创建 XML 对象，存放从数据库获得的记录信息
    var model:XML = XML(EmpList.selectedItem);
    lbl_id.text = model.id;
    txt_name.text = model.name;
    txt_tel.text = model.tel;
    txt_email.text = model.email;
    txt_addr.text = model.addr;
    txt_web.text = model.web;
}
```

(13) 在以上代码添加完毕后，Flex 客户端的通讯录就制作完成了。从工具栏上单击“运行”按钮，生成 SWF 以及对应的 HTML 文件，这些文件位于项目的 bin-debug 目录中。

(14) 将 bin-debug 目录下除 history 子目录之外的文件复制到 ASP.NET 网站 Contacts 的根目录。这一步将 Flex 客户端部署到 ASP.NET 服务器端，为下面的运行做准备。

(15) 最后，在 ASP.NET 网站中检查 SWF 文件、ASPX 文件和 Access 数据库文件，使它们都位于网站的根目录下。

9.4.4 运行结果

至此，利用 Flex+ASP.NET+Access 打造的一个通讯录就大功告成了。

由于 Flex 客户端要访问 ASP.NET，而 ASP.NET 不能在本地运行。因此，需要从 ASP.NET 网站中运行 Contacts.html，页面打开之后会发现只有联系人列表，如图 9-9 所示。

此时，从列表中选择一个联系人，在右侧的 Form 中可查看其详细信息，如图 9-10 所示。



图 9-9 Flex通讯录的运行效果



图 9-10 查看联系人详细信息

9.4.5　实例分析



源码解析：

在本实例中主要演示了 HTTPService 类的使用方法以及如何与 ASP.NET 进行交互。ASP.NET 程序并不复杂，读取数据库返回一个 XML 文档。

重点是在 Flex 中对 HTTPService 各种属性的设置，并创建事件监听函数。send()方法提交了一个请求，disconnect()方法在使用后断开所请求的链接资源。ResultEvent.RESULT 事件将结果绑定到 EmpList 组件。

触发 EmpList 组件的 itemClick 事件后，会获取选中项的所有信息，使得可以查看更多。

如果作为实际应用，还有一些功能需要完善，例如添加一个新联系人、初始化时“删除”按钮不可用、更新和删除成功后弹出提示信息框等。

9.5　Flex与ASP.NET交互的文件上传

虽然 Flex 自带了上传文件的功能，但要实现真正上传功能还必须与 ASP.NET 结合。最基本的过程是：在客户端连接一个主机，服务器接收连接，然后客户端请求一个文件，最后服务器发送一个应答接收文件。



视频教学：光盘/videos/9/FileReference.avi　　　　长度：10 分钟

9.5.1　基础知识——FileReference类

Flex 的文件上传类 FileReference 位于 Flash.net 包下，该类提供了在客户端与服务器之间上传和下载文件的方法，而且还可以获取文件的详细属性，如文件大小、类型、名称、创建日期和修改日期等。

FileReference 类的实例有如下两种创建方法。

(1)　使用 new 运算符调用类的构造函数。

(2)　当调用 FileReferenceList.browse()方法时，返回数组中的每个元素为一个 FileReference 类实例。

FileReference 类主要包含 4 个方法分别用于上传文件、下载文件、取消文件传输、选择上传文件。这 4 个方法如下所示。

- browse()：显示一个文件浏览对话框，让用户选择要上传的文件。
- cancel()：取消正在对该 FileReference 对象执行的任何上传或下载操作。
- download()：打开对话框，以允许用户从远程服务器下载文件。
- upload()：开始将用户选择的文件上传到远程服务器。

在打开对话框，上传和下载文件中，都将触发 FileReference 类中相应的事件。表 9-3 中给

出了常用的事件及说明。

表 9-3　常用事件及说明

事件名称	事件类型	说　明
cancel	Event.CANCEL	当用户通过文件浏览对话框取消文件上传或下载时触发
complete	Event.COMPLETE	当下载操作完成或上传操作生成 HTTP 状态代码 200 时触发
httpStatus	HTTPStatusEvent.HTTP_STATUS	当上传失败并且存在可用来描述失败的 HTTP 状态代码时触发
ioError	IOErrorEvent.IO_ERROR	当上传或下载失败时触发
open	Event.OPEN	当上传或下载操作开始时触发
progress	ProgressEvent.PROGRESS	在文件上传或下载操作期间定期触发
securityError	SecurityErrorEvent.SECURITY_ERROR	尝试将文件上传到调用方安全沙箱外部的服务器，或是从调用方安全沙箱外部的服务器上下载文件时触发
select	Event.SELECT	当用户从文件浏览对话框选择要上传或下载的文件时触发
uploadCompleteData	DataEvent.UPLOAD_COMPLETE_DATA	成功上传并从服务器接收数据之后触发

9.5.2　实例描述

写过很多文件上传的功能，包括 Ajax 实现动态监控上传进度的，现在看到了实现 Flex 文件上传功能。Flex 真是有很多值得学习的地方，而且很方便。

今天就花了点时间研究了一下 Flex 的文件上传，后台采用 ASP.NET 进行处理。实例整合了网上能查找到的一些代码，都是转来转去的，就不在这里列出。还是要感谢前人的分享精神，向他们学习，这里我也和大家分享一下。

9.5.3　实例应用

【例 9-5】实现 Flex 与 ASP.NET 交互的文件上传。

(1) 在 Flex Builder 中打开或新建一个 Flex 项目。本实例在现有的 FileUpload 项目基础上进行修改，主应用程序为 FileUpload.mxml。

(2) 打开主应用程序，然后在合适位置添加文件上传的布局。包括一个 TextInput 组件和两个 Button 组件，代码如下：

```
<mx:HBox width="100%">
   <mx:TextInput id="tipicsrc" editable="false" width="310"/>
   <mx:Button label="浏览" click="BrowsePicFiles()"/>
```

```
    <mx:Button id="btnUploadpic" label="上传" click="UploadPic()"
     enabled="false"/>
</mx:HBox>
```

(3) 为了使文件上传时更加直观。这里添加了一个 ProgressBar 组件来作为上传进度条：

```
<mx:ProgressBar id="progressbar" labelPlacement="center" trackHeight="15"
  height="20" color="#000000" width="480"/>
```

(4) 经过上面两步，上传的布局就完成了。下面开始编写 Flex 端的上传代码，第一步先在<mx:Script>标记中添加文件上传所需的引用和变量，具体如下：

```
<mx:Script>
<![CDATA[
    import mx.events.CloseEvent;            //添加所需的引用
    import mx.controls.DateChooser;
    import mx.rpc.events.FaultEvent;
    import mx.rpc.events.ResultEvent;
    import mx.controls.Alert;
    import Flash.net.*;
    private var  refUploadFile1:FileReference; //声明所需的变量
    private var _arrUploadFiles:Array;
    private var _filefilter:Array;
    //允许上传的图片类型
    private var imageTypes:FileFilter = new FileFilter("Images (*.jpg; *.jpeg;
      *.gif; *.png)" ,"*.jpg; *.jpeg; *.gif; *.png");
    //指定服务器端接收文件的名称
    private var _URL:String = "Server/FileServer.aspx";
    private var  uploadURL:URLRequest;
]]>
</mx:Script>
```

(5) “浏览”按钮单击后会弹出一个对话框，让用户选择需要上传的文件，同时在这里可以指定允许选择文件的类型。具体代码如下所示：

```
//浏览图片
private function BrowsePicFiles():void {
    _refUploadFile1 = new FileReference();
    _refUploadFile1.addEventListener(Event.SELECT, onSelectPicHandler);
    _refUploadFile1.addEventListener(Event.COMPLETE, onUploadComplete);
    _refUploadFile1.addEventListener(
      IOErrorEvent.IO_ERROR, onUploadIoError);
    _refUploadFile1.addEventListener(
      ProgressEvent.PROGRESS, onUploadProgress);
    _refUploadFile1.addEventListener(
      SecurityErrorEvent.SECURITY_ERROR, onUploadSecurityError);
    _filefilter = new Array(imageTypes);
    _refUploadFile1.browse(_filefilter); //允许选择的文件类型
}
```

如上述代码所示，主要是通过 FileReference 类来完成，同时对该类在上传过程中的各个事

件设立了监听函数。

(6) 在弹出的对话框中选择一个文件，触发 Event.SELECT 事件并转到 onSelectPicHandler()函数执行代码。在该函数中显示选择文件的名称并使“上传”按钮可用，具体代码如下：

```
//选择图片文件
private function onSelectPicHandler(event:Event):void {
    this.tipicsrc.text = _refUploadFile1.name;       //显示选择的文件名称
    this.btnUploadpic.enabled = true;                //启用“上传”按钮
}
```

(7) 选择文件之后，单击“上传”按钮提交上传请求。其单击事件代码如下：

```
//上传图片
private function UploadPic():void
{
    _uploadURL = new URLRequest;
    _uploadURL.url = _URL;
    _uploadURL.method = "GET";
     uploadURL.contentType = "multipart/form-data";
    _refUploadFile1.upload(_uploadURL);
}
```

(8) 上述代码中调用了 upload()方法向_URL 指定的文件中输送文件流。此时，将会由 ProgressEvent.PROGRESS 事件的处理函数 onUploadProgress()显示上传进度。该函数的实现代码如下：

```
//上传过程中
private function onUploadProgress(event:ProgressEvent):void {
    progressbar.setProgress(event.bytesLoaded,event.bytesTotal);
    progressbar.label = "Uploading " + Math.round(event.bytesLoaded / 1024)
      + " kb of " + Math.round(event.bytesTotal / 1024) + " kb " ;
}
```

(9) 上传完成时，将会显示“Uploads Complete”并弹出对话框，这是由 Event.COMPLETE 事件的监听函数 onUploadComplete()完成的：

```
//上传完成
private function onUploadComplete(event:Event):void {
    trace("uploadcomplete");
    Alert.show("所选文件上传成功。", "添加");
    progressbar.label = "Uploads Complete";
}
```

(10) 为了更好地处理文件上传过程，还需要监听两个出错事件。它们由 onUploadIoError()函数和 onUploadSecurityError()函数进行处理，代码如下：

```
//IO 导致的上传失败
private function onUploadIoError(event:IOErrorEvent):void {
    Alert.show(String(event), "ioError", 0);
}
//权限导致的上传失败
```

```
private function onUploadSecurityError(event:SecurityErrorEvent):void {
    Alert.show(String(event), "Security Error", 0);
}
```

(11) 从工具栏上单击“运行”按钮生成 SWF 以及对应的 HTML 文件。此时，会发现程序虽然能运行，但无法实现上传功能。因为，真正把文件上传到服务器上是靠 ASP.NET 完成的。

(12) 将生成的文件复制到 ASP.NET 网站的根目录下。然后，在 Server 子目录下新建 FileServer.aspx 文件接收客户端的上传请求。最终代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    HttpFileCollection uploadedFiles = Request.Files;
    string Path = Server.MapPath("~/uploadFiles/");
    // Response.Write(Path);
    for (int i=0; i<uploadedFiles.Count; i++)
    {
        HttpPostedFile F = uploadedFiles[i];
        if (uploadedFiles[i]!=null && F.ContentLength>0)
        {
            string newName =
              F.FileName.Substring(F.FileName.LastIndexOf("\\") + 1);
            F.SaveAs(Path + "/" + newName);
        }
    }
}
```

至此，程序的编码工作就完成了。但离运行还差一步，就是需要在网站中创建 uploadFiles 目录作为文件上传的根目录。

9.5.4　运行结果

从 ASP.NET 网站中打开包含 SWF 的文件。然后，单击“浏览”按钮并选择一个文件，此时会看到选择的文件名称，“上传”按钮也变得可用，如图 9-11 所示。



图 9-11　选择文件

之后，单击“上传”按钮提交，在下方将显示上传进度。上传完成后将弹出对话框提示，如图 9-12 所示。



图 9-12　文件上传成功

9.5.5　实例分析



源码解析：

通过本实例的制作过程，我们得出如下结论：

实现文件上传即客户端向服务器发送一个请求，再把请求中的信息封装在 HTTP 协议的请求头中。然后服务端读取请求头中的信息，之后用流的方式写到指定的位置。

而在整个过程中，Flex 中的 FileReference 类起到了重要作用，可以指定上传类型，获取上传大小，指定请求的名称以及处理异常等。

9.6　常见问题解答

9.6.1　ASP.NET中向Flash传递XML对象



ASP.NET 中向 Flash 传递 XML 对象？

网络课堂：http://bbs.itzcn.com/thread-9861-1-1.html

数据库中有很多 XML 文件路径(如 a.xml、b.xm、...)，我希望根据要求从数据库中得到 XML 路径，然后传递给 flash。这样 flash 可以通过 onload 加载一个动态的路径加载不同的 XML，而不是一个固定的路径。我只知道可以使用<%Response.ContentType =“text/XML” %>。具体如何做，由于是初学，所以不知道。

【解决办法】：在 Flash 中创建一个 XML 包。XML 数据包能容易被创建在 Flash 中，如下所示：

```
var sendXML:XML = new XML("<userid>123456</userid>"=;
```

发送一个 XML 包到一个服务器端页面——它使用 ASP/ASP.NET 并且处理返回 XML 数据包。用法如下:

```
xmlObject.sendAndLoad(URL:String, targetXMLObject:XML):Void
```

这个 xmlObject 是 XML 对象实例，它封装必须被发送到.aspx 页面的 XML；这个 URL 是指定 XML 对象的目的 URL。

targetXMLObject 是该 XML 对象，它将接收从服务器端返回的信息。

sendAndLoad 方法把一个 XML 包发送到一个服务器端模板并且收到一个 XML 包作为响应。这不同于 XML 类的 send 方法，它仅发送一个 XML 包而并不期盼任何类型的响应。这其中每个方法以其自身方式发挥作用，但是有一重要的区别：XML 类的 send 方法以一个目标作为参数，它允许你打开一个新的浏览器窗口或替换当前浏览器窗口中的内容，而 sendAndLoad 方法以一个 XML 包作为参数，它不会启动一个新的浏览器窗口。因为 send 方法打开一新的浏览器窗口，我们可以看到浏览器窗口本身的输出。这可能对调试极为有用。

9.6.2　ASP.NET从数据库中读取Flash广告



ASP.NET 如何从数据库中读取 Flash 广告？
网络课堂：http://bbs.itzcn.com/thread-9862-1-1.html

情况是这样的：我找了一个广告轮换的 Flash 代码，但是广告的内容和描述都是放到了一个 XML 文件里，然后再从 HTML 中读取 XML 里的数据。

但是我是用 ASP.NET(C#)做的网站，想从数据库中读取广告的图片和描述内容，可是我只会读取到页面的前台上，不知道怎样把数据放到那个 XML 文件里。

【解决办法】：那个 XML 不要用静态的，用一个 ashx 来输出 XML 格式的文件，然后 Flash 读取那个 ashx。

Handler.ashx 页面的实现代码如下：

```
<%@ WebHandler Language="C#" Class="Handler" %>
using System;
using System.Web;
using System.Linq;
using System.Collections.Generic;
public class Handler : IHttpHandler
{
    public void ProcessRequest(HttpContext context)
    {
        context.Response.ContentType = "text/XML";
        //循环要输出的数据
        for (int i=0; i<50; i++)
        {
            //循环输出数据文字到 XML 格式，写入 Response 流
            context.Response.Write();
```

```
        }
    }
    public bool IsReusable
    {
        get
        {
            return false;
        }
    }
}
```

9.7 习　题

一、填空题

(1) 在 ActionScript 2.0 中，Flash 应用程序要与服务器之间传输变量，需要用到________类。

(2) ________的主要功能是在包含 Flash Player 容器的应用程序(通常是一个浏览器)中，打开或替换一个窗口。

(3) 在 XML 类中，________属性用于计算指定的 XML 对象，并引用父级节点的子级列表中的第一个子级。如果节点没有子级，则此属性为________。

(4) 在 Flex 中采用________交互方式时，传递的数据类型可以是数组、文件或者图片。

(5) 使用 HttpService 方式交互时，需要在________事件的监听函数中处理返回结果。

(6) 位于________包下 FileReference 类提供了在客户端与服务器之间上传文件的方法。

二、选择题

(1) 下列选项中，哪一个不是 LoadVars 的方法________。

A. onLoad()　　B. send()　　C. onSend()　　D. sendAndLoad()

(2) 有一个实例名为“Msg”变量名为“msg”的动态文本框，下面________语句能够为该动态文本框赋值。

A. Msg.text=“Hello World!”;　　B. Msg=“Hello World!”;

C. msg.text=“Hello World!”;　　D. msg=“Hello World!”;

(3) 在 ActionScript 3.0 中，________类以文本、二进制数据或 URL 编码变量的形式从 URL 下载数据。

A. URLStream　　B. URLLoader

C. URLVariables　　D. URLRequest

(4) 在如下 XML 文档中，下列________语句能正确获得一个 XML 文档子节点属性 name 的值。

```
<root>
    <cd name="软件开发">
        <caidan subname="C#从入门到精通" url="#"/>
```

```
    </cd>
</root>
```

A. myxml.firstChild.childNodes[1].attributes.name;

B. myxml.firstChild.childNodes[1].nodeValue;

C. myxml.firstChild.childNodes[1].childNodes;

D. myxml.firstChild.childNodes[1].nodeName;

(5) 下列描述 HttpService 交互方式的语句中，错误的是__________。

A. 它采用 HTTP 协议进行通讯

B. 数据格式通用，便于不同应用系统间交换数据

C. 它不适合处理复杂数据类型

D. 它能够处理各种类型的数据，速度快

(6) _refUploadFile 是一个 FileReference 类的实例，在下面的空白处填写代码__________可以监听上传的进度。

```
_refUploadFile.addEventListener(__________, onUploadComplete);
```

A. Event.COMPLETE

B. ProgressEvent.PROGRESS

C. SecurityErrorEvent.SECURITY_ERROR

D. IOErrorEvent.IO_ERROR

三、上机练习

上机练习 1：用 Flash 制作一个登录界面。

本次练习要求读者制作一个 Flash 登录界面并实现登录和简单的登录验证功能。用户登录成功后，在 Flash 界面显示友好的登录消息，效果如图 9-13 所示。本实例的目的在于让读者熟悉一下 LoadVars 类与服务器端通信的机制。

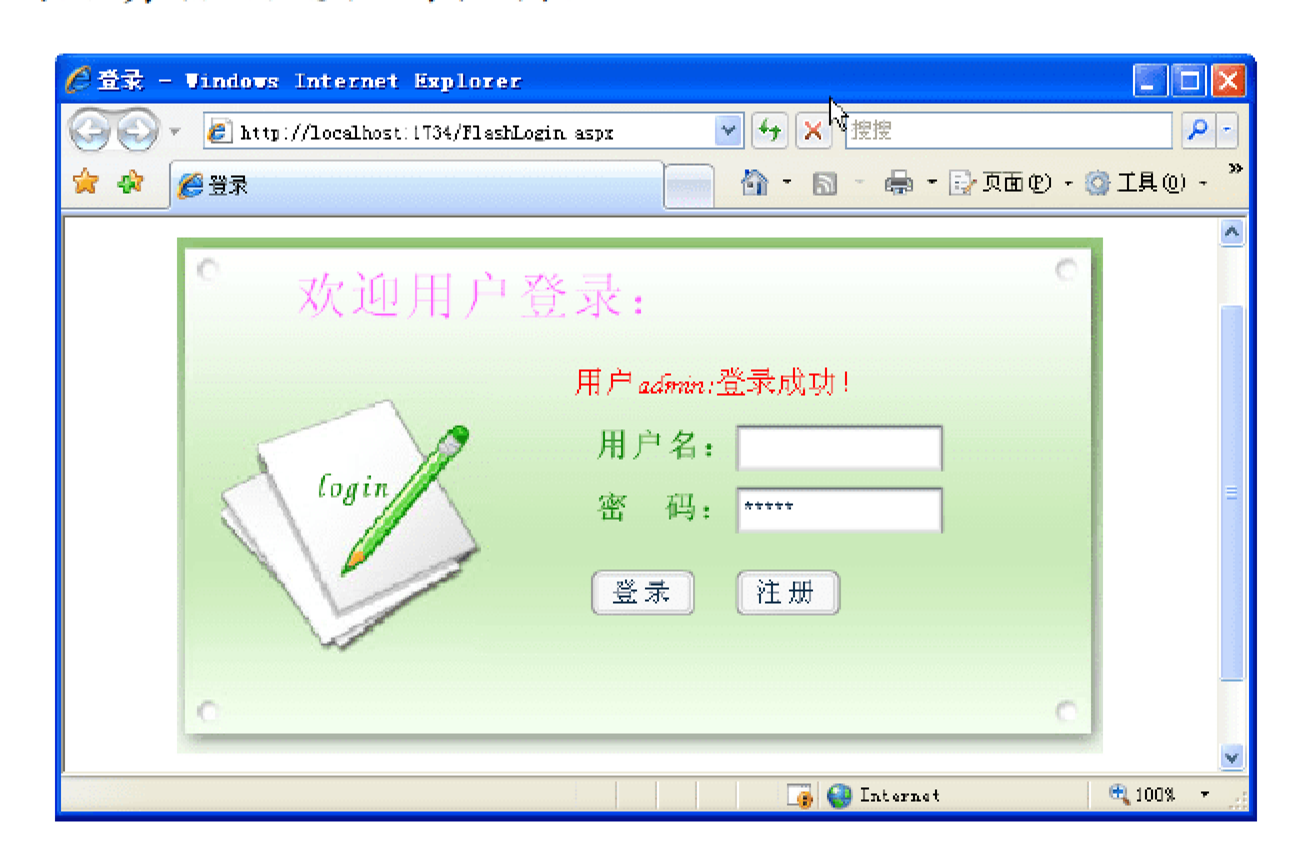


图 9-13　Flash登录界面的运行效果

上机练习 2：Flex 获取图书列表。

前面我们学习了如何使用 HttpService 方式进行数据交互。本次练习需要读者使用 Flex 完成，重点是数据交互的过程及处理方式。

这是一个保存了图书信息的列表，通过 ASP.NET 读取并返回格式良好的 XML。Flex 中利用 HttpService 获取 XML，并在监听事件中绑定数据，最后显示出来，运行效果如图 9-14 所示。



图 9-14　图书列表运行效果



第 10 章　控制页面颜色与绘图

内容摘要：

在科技发达的今天，医学整容蔚然成风：“我想要巩俐的鼻子，林青霞的下巴”。

这说明“以貌取人”事件越来越多地发生在我们的身边。

在计算机的发展史上，Windows 操作系统的出现，即以“外观”漂亮、用户友好等优势，受到越来越多用户的欢迎和喜爱。

而在 Windows 操作系统“漂亮外观”的背后，都离不开一位功臣，那就是 GDI。

我们利用 GDI 所提供的众多函数，可以方便地在屏幕、打印机及其他输出设备上做输出图形、文本等操作。

随着 Windows 的发展，作为 GDI 的继承者——GDI+不但完全兼容旧版本，而且提供了很多新功能，来满足图形输出的需要。

在本章中，我们将重点对 ASP.NET 下如何利用 GDI+控制页面的颜色和在页面上绘图进行介绍。例如，生成随机验证码、对图片进行缩放、添加水印等。

学习目标：

- 了解.NET 下的 GDI+
- 掌握 GDI+中坐标和颜色的控制
- 掌握如何使用画刷填充指定区域
- 掌握在页面显示文本的方法
- 掌握将图像显示到页面的方法
- 掌握如何绘制柱状图表
- 掌握如何绘制饼状图表

10.1 如何使用.NET进行绘图

对于 Windows 系统而言，它有一个优点：即开发人员不必考虑特定设备的细节。例如，不需要理解硬盘设备驱动程序，只需在相关的.NET 类中调用合适的方法，就可以在程序中读写磁盘上的文件。

与此类似，计算机在屏幕上绘图时，只需要把指令发送给视频卡即可。问题是市场上有几百种不同的视频卡，大多数都有不同的指令集和功能。如果将该因素考虑在内，则在应用程序中就需要为每个视频卡驱动程序编写在屏幕上绘图的特定代码，这样的应用程序就根本不可能编写出来。这就是为什么在 Windows 最早期的版本中就有 Windows Graphical Device Interface (GDI)的原因。

GDI+提供了一个抽象层，隐藏了不同视频卡之间的区别，这样就可以只调用 Windows API 函数即可完成特定的任务。GDI+会在内部根据不同的视频卡使用不同的代码，让客户机的视频卡完成要绘制的图形。

GDI+还可以完成其他任务。大多数计算机都有多个显示设备——例如，监视器和打印机。GDI+成功地使应用程序所使用的打印机看起来与屏幕一样。如果要打印某些东西，而不是显示它们，只需告诉系统输出的设备是打印机，再用相同的方式调用相同的 Windows API 函数即可。表 10-1 列出了 GDI+中的主要命名空间。

表 10-1　GDI+命名空间

命名空间	说　明
System.Drawing	包含与基本绘图功能有关的大多数类、结构、枚举和委托
System.Drawing.Drawing2D	为大多数高级 2D 和矢量绘图操作提供了支持，包括消除锯齿、几何转换和图形路径
System.Drawing.Imaging	帮助处理图像(位图、GIF 文件等)的各种类
System.Drawing.Printing	把打印机或打印预览窗口作为输出设备时使用的类
System.Drawing.Design	一些预定义的对话框、属性表和其他用户界面元素，与在设计期间扩展用户界面相关
System.Drawing.Text	对字体和字体系列执行更高级操作的类

10.2 绘图基本功

本节中的内容是对 GDI+的概述，初学者不要期望一下子就要很好地理解它们，而应在学习了本章后面的内容后再回过头来细细揣摩，反复多次，并注重 GDI+的编程实践，这样就一定能够很好地掌握 GDI+，并能够在编程实践中自如地使用 GDI+。

10.2.1　GDI+坐标系统

GDI+的坐标系统与数学中的坐标系统不同，它不是以中心为原点，而是以屏幕的左上角为坐标原点，表示为(0, 0)。用于绘图的每个窗口都有自己的坐标系统，坐标原点为窗口的左上角，这样，在窗体中绘制的控件和图形的坐标都是相对于窗口的，不需要处理窗口在屏幕上的位置。

例如，从点(1, 1)到点(5, 1)画一条具有一个像素宽的线段，结果如图 10-1 所示。

从点(2, 1)到点(2, 4)画一条具有一个像素宽、4 个像素长的垂直线段，结果如图 10-2 所示。

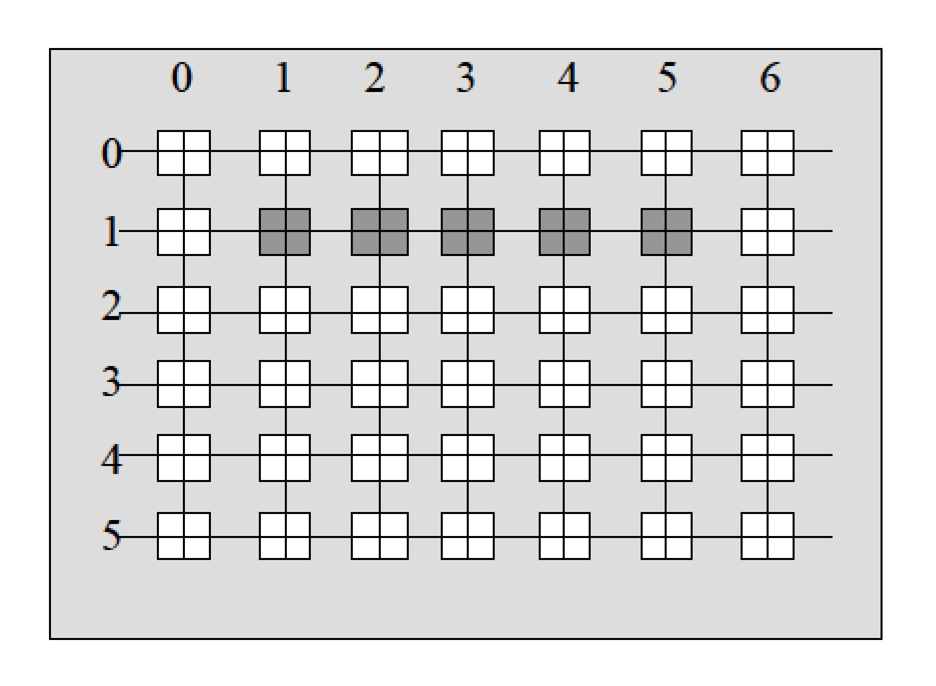


图 10-1　绘制水平线

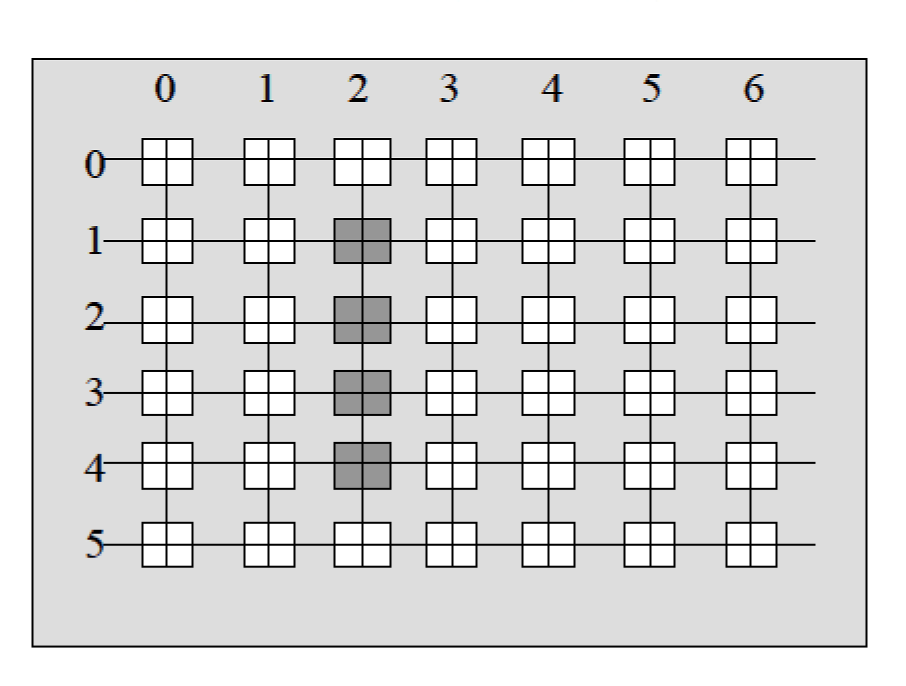


图 10-2　绘制垂直线

绘制从点(1, 0)到点(4, 3)的对角线，结果如图 10-3 所示。绘制一个左上角点为(1, 0)，右下角点为(6, 4)的矩形，结果如图 10-4 所示。

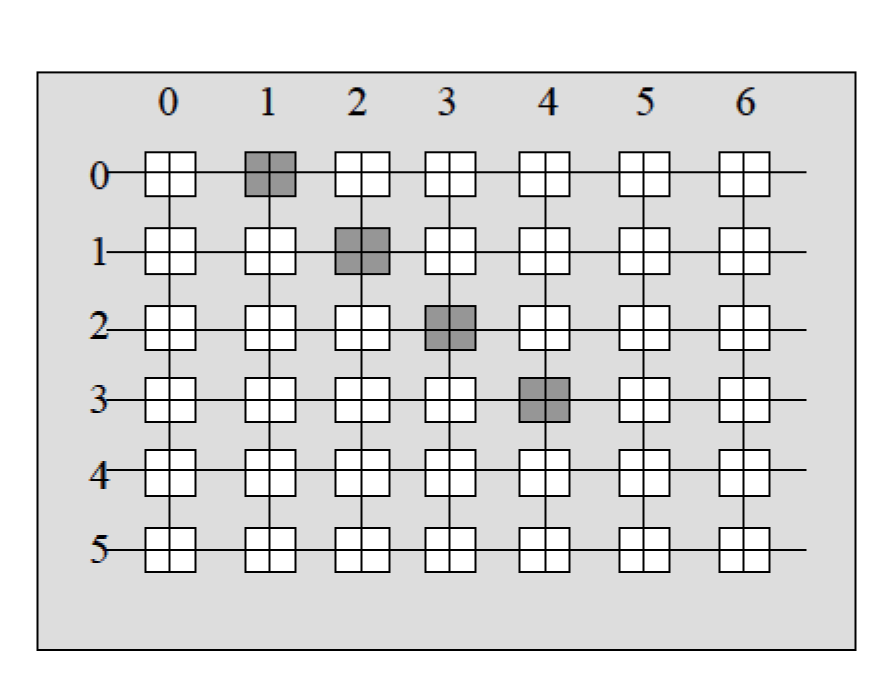


图 10-3　绘制斜线

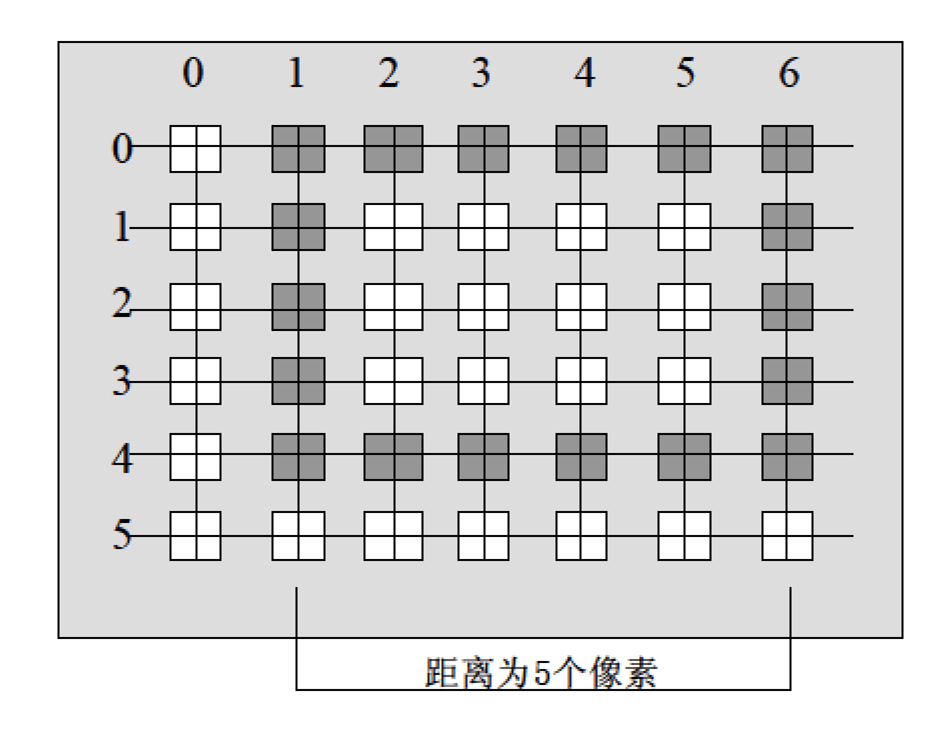


图 10-4　绘制矩形

10.2.2　GDI+坐标结构

在理解了绘图坐标系统之间的关系后，通过观察绘图界面上的结果，很容易确定哪些像素受到绘图效果的影响。在绘图时，常常使用 Point、Size 和 Rectangle 这 3 种结构指定坐标。

1. Point结构

Point 结构表示二维平面上的一个点或者像素。Graphics 类中的许多绘图方法如 DrawLine()

等都以 Point 结构的变量作为参数。可像如下所示那样声明和构造 Point 结构的变量：

```
Point p = new Point(2, 5);
```

该行代码定义了一个 Point 结构变量 p，它代表绘图面上横坐标为 2，纵坐标为 5 的像素(点)。还有一种是 PointF 结构，它与 Point 结构类似，唯一不同的是可以使用浮点定义坐标。

2. Size结构

System.Drawing 命名空间的 Size 结构存储一个有序整数对，通常为矩形的宽度和高度。Size 结构构造函数的两种形式为：

```
Size(Point pt);
Size(int width, int height);
```

例如，可使用如下所示的代码定义 Size 结构的变量：

```
Size s = new Size(5, 10);
```

其中，5 为 Size 结构的水平分量；10 为 Size 结构的垂直分量。

3. Rectangle结构

Rectangle 结构存储 4 个整数，用于表示矩形的位置和大小。Rectangle 结构具有两个构造函数：

```
Rectangle(Point location, Size size);
Rectangle(int x, int y, int width, int height);
```

其中，location 用于指定矩形区域的左上角点坐标，size 用于指定矩形区域的宽度和高度；X 用于指定矩形左上角点的 X 坐标，Y 用于指定矩形左上角点的 Y 坐标，width 用于指定矩形区域的宽度，height 用于指定矩形区域的高度。

可按如下所示那样定义 Rectangle 结构的变量：

```
Rectangle r1 = new Rectangle(5, 5, 10, 15);
//变量 r1 所代表的矩形区域的左上角点为(5, 5)，矩形区域的宽度为 10，高度为 15

Point pt = new Point(5, 5);
Size size = new Size(10, 15);
Rectangle r2 = new Rectangle(pt, size);
//这里的 r2 所代表的矩形区域的位置和大小与 r1 是完全一样的
```

10.2.3 GDI+颜色体系

在 GDI+中，使用 Color 结构来表示颜色；其属性有 A、R、G、B，分别表示 alpha、红、绿、蓝分量值，它们的取值范围是 0~255。

Color 结构中包含很多使用静态属性表示的预定义颜色，在引用它们时，需要在它们的前面加前缀“Color.”，例如，Color.Green 用于表示预定义的绿色。

要使用颜色，只需声明一个 Color 类型的变量，然后用 Color 结构中的一种颜色初始化该变量。如下所示：

```
Color redColor = Color.Red;
Color yellowColor = Color.Yellow;
```

在创建了 Color 结构实例之后，就可以将其作为参数传递给需要使用颜色的绘图方法，然后调用该方法，就可以用 Color 结构实例所代表的颜色进行绘图了。当然，也可以不创建 Color 结构变量，直接使用 Color 结构中用于表示颜色的属性。

除此之外，还可以使用 Color.FromArgb()方法创建用户自定义颜色，该方法要求必须指定一种颜色中红色、绿色和蓝色各部分的值。该方法的几种重载形式为：

```
static Color FromArgb(int alpha);
static Color FromArgb(int alpha, Color color);
static Color FromArgb(int r, int g, int b);
static Color FromArgb(int alpha, int r, int g, int b);
```

具体的示例代码如下：

```
Color myColor1;
myColor1 = Color.FromArgb(122, 80, 180, 20);
```

10.3　System.Drawing命名空间

System.Drawing 命名空间提供了对 GDI+基本图形功能的访问，它提供了很多的类，用于实现强大的图形处理功能。例如，Graphics 类提供了绘制到显示设备的方法；Rectangle 和 Point 等类可封装 GDI+位图；Pen 类用于绘制直线和曲线，而从抽象类 Brush 派生出的类则用于填充形状的内部。在表 10-2 中列出了该命名空间常用的类。

表 10-2　System.Drawing命名空间常用的类

类	说　明
Bitmap	封装 GDI+位图，该位图由图形图像及其属性的像素数据组成，还提供一些方法保存位图
Graphics	封装一个 GDI+绘图画布，这是位图上画图的主要工具，提供将对象绘制到显示设备的方法
Icon	表示 Windows 图标，是用于表示对象的小位图图像。尽管图标的大小由系统决定，但仍可将其视为透明的位图
Pen	用于画直线和曲线，可以指定画笔的属性
Pens	表示所有标准颜色的钢笔
Brush	用于填充形状和画图文本。用作基类，可以以其派生类来定义可填充的图形对象，如椭圆、矩形多边形等
Brushes	表示所有标准颜色的画笔
Font	为文本定义字体属性，包括字体类型、大小、样式等
PointConverter	将 Point 对象从一种数据类型转换为另一种数据类型

续表

类	说　明
Image	用于支持位图、指针、图标以及图形文件的类
RectangleConverter	将矩形从一种数据类型转换为另一种数据类型。通过 TypeDescriptor 访问此类
StringFormat	封装文本布局信息(如对齐、文字方向和 Tab 停靠位)，显示操作和 OpenType 功能
SystemColors	SystemColors 类的每个属性都是 Color 结构，这种结构是 Windows 显示元素的颜色
TextureBrush	TextureBrush 类的每个属性都是 Brush 对象，这种对象使用图像来填充形状的内部

10.4　动态绘制公司Logo图片

使用 System.Drawing 命名空间提供的类实现绘图的原理非常简单。Bitmap 就像一张画布，Graphics 如同画布的手，把 Pen 或 Brush 等绘图工具在 Bitmap 这张画布上画出来。

本节我们将动态打造一个图文并茂的公司 Logo 图片，以演示图形绘制的基础知识和基本技术。

 **视频教学：**光盘/videos/10/Graphics.avi　　　　**长度：7 分钟**

10.4.1　基础知识——Graphics类

在 GDI+中，大多数绘图工作都是调用.NET 基类 System.Drawing.Graphics 实例的方法完成的。实际上，因为 Graphics 类负责处理大多数绘图操作，所以 GDI+中很少有操作不涉及到 Graphics 实例。

既然如此重要，那么理解如何处理这个对象是理解如何使用 GDI+在页面上绘图的关键。

由于 Graphics 类没有定义构造函数，因此无法使用 new 关键字的方法创建 Graphics 对象。但如果要使用 GDI+绘制线条和形状、显示文本或者图像，就先必须先创建 Graphics 对象。

在 ASP.NET 中通常可以通过调用该类的 FromImage()方法来创建一个 Graphics 对象。FromImage()方法支持的是 Image 类型或者其他派生类型。

例如，下面的代码演示了这种方法的两种创建形式：

```
Bitmap bitmap1 = new Bitmap("background.jpg");
Graphics g1 = Graphics.FromImage(bitmap1); //从外部图像中创建 Graphics 对象

Bitmap bitmap2 = new Bitmap(100, 100);
Graphics g2 = Graphics.FromImage(bitmap2); //从指定区域中创建 Graphics 对象
```

与先前章节中相同，Graphics 类中具有许多属性和方法，熟练使用这些属性和方法将有助于 ASP.NET 的开发工作，这里来介绍常用属性及其用途。表 10-3 显示了 Graphics 类中常用的属性。

表 10-3　Graphics类中的常用属性

属　性	说　明
Clip	获取或设置 Region，该对象限定此 Graphics 的绘图区域
ClipBounds	获取一个 RectangleF 结构，该结构限定此 Graphics 的剪辑区域
DpiX	获取此 Graphics 的水平分辨率
DpiY	获取此 Graphics 的垂直分辨率
IsClipEmpty	获取一个值，该值指示此 Graphics 的剪辑区域是否为空
IsVisibleClipEmpty	获取一个值，该值指示此 Graphics 的可见剪辑区域是否为空
PageScale	获取或设置此 Graphics 的全局单位和页单位之间的比例
PageUnit	获取或设置用于此 Graphics 中的页坐标的度量单位
PixelOffsetMode	获取或设置一个值，该值指定在呈现 Graphics 的过程中像素如何偏移
SmoothingMode	获取或设置此 Graphics 的呈现质量
TextContrast	获取或设置呈现文本的灰度校正值
TextRenderingHint	获取或设置与此 Graphics 关联的文本的呈现模式
Transform	获取或设置此 Graphics 的几何世界变换的副本
VisibleClipBounds	获取此 Graphics 的可见剪辑区域的边框

10.4.2　实例描述

在内心世界中，感觉 GDI+是个非常神秘的东西，一直不敢去深入了解和接触。但是直到昨天，真正得去面对它了。

昨天经理说了一句话，如果你不把它给搞定了，干到今天就 over 了。不想被 over 掉啊，于是就必须把 GDI+给搞定！这是啥子世道啊。

现在是深有感触的时候。在工作中不能有“我不会，我不清楚”，有的只是自己在工作中的难和苦。自己的苦只有自己吃！

唠叨了这么多了，就准备开始搞定 GDI+了！下面先拿公司的 Logo 图片开刀……

10.4.3　实例应用

【例 10-1】动态绘制公司 Logo 图片。

我们公司的 Logo 还是传统的老掉牙的那一种，左边一个图片，右边是公司名称，给人的感觉很简洁，也没有过多的背景或者花纹装饰，很像大牌 Logo。下面来看一下我画的总体布局与结构吧，如图 10-5 所示。

打开早已再熟悉不过的 VS2008，新建一个名为 logo.aspx 的文件。删除掉第 1 行之外的多余代码。之后在.cs 文件中引用 GDI+绘图所需的 3 个命名空间：

```
using System.Drawing;
using System.Drawing.Drawing2D;
```

```
using System.Drawing.Imaging;
```

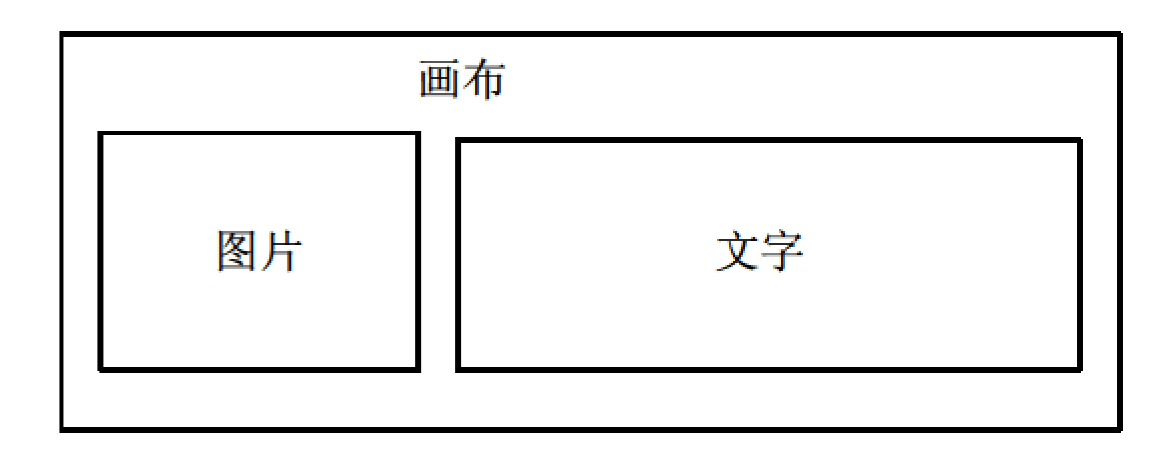


图 10-5　图片的布局与结构

接下来进入页面的 Page_Load()方法，在这里开始我们的绘画创作。先写上一句声明图片格式为 gif 的代码：

```
//设置请求格式为 gif
Response.ContentType = "image/gif";

//然后指定可用于我们创作的空间大小，并对空间进行简单修饰。如划出一块区域并填充颜色为下一步
//的画图做准备。

//指定画布大小
Bitmap bitmap = new Bitmap(240, 50);
//创建画布
Graphics g = Graphics.FromImage(bitmap);
//设置画布的属性
g.InterpolationMode = InterpolationMode.HighQualityBicubic;
g.CompositingQuality = CompositingQuality.HighQuality;
//文字抗锯齿
g.TextRenderingHint = System.Drawing.Text.TextRenderingHint.AntiAlias;
//使用白色填充画布
g.FillRectangle(Brushes.White, 1, 1, bitmap.Width - 2, bitmap.Height - 2);
```

从设计人员那里搞到公司的 logo，放在站点的根目录下。根据草图的设计，首先我们需要将这个图片放到左边：

```
//从外部文件创建一个图片
Image image = Bitmap.FromFile(Server.MapPath("logo.gif"));
//绘制图片
g.DrawImage(image, 5, 5);
```

为图片的右边添加公司名称，当然还要设置字体的大小和颜色等，使它尽量显示得与图片协调：

```
//创建一个笔刷
SolidBrush sbCurrent = new SolidBrush(Color.FromArgb(23, 157, 244));
//创建一种字体
Font font = new Font("幼圆", 20, FontStyle.Bold);
//使用笔刷和字体绘制文本
g.DrawString("汇智科技", font, sbCurrent, 110, 15);
```

至此，一个图文混排的 Logo 就出来了。将上面的工作成果发送到页面的输出流中进行显示：

```
//创建一个输出流
System.IO.MemoryStream ms = new System.IO.MemoryStream();
//将画布的内容输出到流
bitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);
//输出流
Response.ClearContent();
Response.BinaryWrite(ms.ToArray());
```

最后，清理一下绘图工作台，将不用的都丢掉：

```
//释放资源
bitmap.Dispose();
g.Dispose();
Response.End();
```

离成功只差一步。打开公司的博客首页，使用下面的一行代替原来的 img 标记。

```
<img src="logo.aspx" />
```

10.4.4　运行结果

将页面保存好，在浏览器中直接运行 logo.aspx，将会看到绘制的图片。而如果从主页上运行，将会看到非常不错的整体效果，如图 10-6 所示。



图 10-6　动态绘制公司Logo图片的效果

10.4.5　实例分析



源码解析：

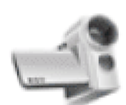
作为第一个 GDI+作品，这个效果感觉非常棒。

Graphics 类是绘图时的画布，Graphics.FromImage(bitmap)语句从 bitmap 中创建一个画布。

再往下的绘图 DrawImage()和绘制文本 DrawString()均是使用该类提供的方法在画布上进行操作。语句“bitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg)”将画布保存到 ms 流中。绘制完毕之后，“Response.BinaryWrite(ms.ToArray())”语句将画布以流的形式输出到页面中进行显示。最后提醒一点：执行绘图操作可能会占用大量的内存资源，如果内存不足，应用程序就不能正确绘制。因此，在本实例最后调用对象的 Dispose()方法删除对象并释放对象所占用的资源。

10.5 实现验证码

在 Web 系统中，很多时候需要用到验证码。例如我们经常遇到不少电子邮件、论坛的注册过程需要输入验证码，这是为了提高安全性。今天我们就来讲讲如何生成验证码。

 **视频教学：**光盘/videos/10/Brush.avi　　长度：5 分钟

10.5.1 基础知识——MeasureString()方法

简单来说，Graphics 类的 MeasureString()方法就是用来在 GDI+中获取字符串实际的输出像素值，包括高度和宽度。

下面是一个简单的例子：

```
String text = "水印";
Font myFont = new Font("宋体", 14);
SizeF size = g.MeasureString(text, myFont);
```

MeasureString()方法常用的重载形式有如下几种：

```
MeasureString(String text, Font font);
MeasureString(String text, Font font, Int32 width);
MeasureString(String text, Font font, SizeF layoutArea);
MeasureString(String text, Font font, SizeF layoutArea, StringFormat
  stringFormat, Int32 charactersFitted, Int32 linesFilled);
```

方法会返回一个 SizeF 结构，该结构表示 text 参数指定的、使用 font 参数绘制的字符串的大小，单位由 PageUnit 属性指定。上述重载形式中各参数的含义如下。

- text：要测量的字符串。
- font：它定义字符串的文本格式。
- width：字符串的最大宽度(以像素为单位)。
- layoutArea：它指定文本的最大布局区域。
- stringFormat：它表示字符串的格式化信息(如行距)。
- charactersFitted：字符串中的字符数。
- linesFilled：字符串中的文本行数。



GetHeight()获得的高度跟 MeasureString()获得的高度是不同的。GetHeight()获得的高度是该字体刚好的像素高度，而 MeasureString()获得的高度是包含该字符串的区域的高度。

10.5.2　基础知识——画刷

画刷是一种用来填充区域的工具，可以是矩形、椭圆或者多边形等。在 GDI+中，Brush 基类封装了画刷的基本功能，由它又派生出 SolidBrush 类、TextureBrush 类、HatchBrush 类和 LinearGradientBrush 等类，它们分别用于单色画刷、纹理画刷、阴影画刷和线性渐变画刷。其中，Brush 和 SolidBrush 类在 System.Drawing 命名空间中，而 TextureBrush 和 LinearGradinetBrush 类在 System.Drawing.Drawing2D 命名空间中。

1. 单色画刷

单色画刷是最简单的一种画刷，可用于填充图形形状，例如矩形、椭圆、扇形、多边形和封闭路径，由 SolidBrush 类实现。要使用单色画刷，只需要在实例化时为 SolidBrush 类的构造函数指定一个颜色值。例如：

```
SolidBrush sbCurrent = new SolidBrush(Color.Black);     //创建一个黑色画刷
Rectangle recCurrent = new Rectangle(10, 10, 500, 300);     //创建一个矩形
Graphics.FillRectangle(sbCurrent, recCurrent);     //使用画刷填充矩形
sbCurrent.Dispose();                                //销毁自定义画刷
```

这里调用 Graphics 类的 FillRectangle()方法进行填充，两个参数分别是画刷对象和要填充的图形。它还可以和其他类型的画刷填充矩形，方法具有两个参数。

还可以使用系统预定义的画刷。系统预定义的画刷由 System.Drawing.Brushes 类封装，不需要显式销毁。例如，使用系统预定义的黄色画刷填充矩形：

```
Rectangle recCurrent = new Rectangle(10, 10, 200, 200);     //定义区域
Graphics.FillRectangle(Brushes.Yellow, recCurrent);         //填充区域
```

2. 阴影画刷

阴影画刷是一种由阴影样式、前景色和背景色属性构成的画刷。要使用阴影画刷填充矩形区域或其他区域，需要先创建一个 HatchBrush 类的实例，然后将该实例作为填充区域的方法(例如用于填充矩形的 FillRectangle 方法)的实例。HatchBrush 类的构造函数如下：

```
public HatchBrush(HatchStyle hatchstyle, Color foreColor)
public HatchBrush(HatchStyle hatchstyle, Color foreColor,Color backColor)
```

在上述构造函数中：

- Hatchstyle——指定阴影画刷的阴影样式，是 System.Drawing.Drawing2D.HatchStyle 枚举值之一。
- forecolor 和 backcolor——指定阴影画刷的前景色和背景色。前景色是阴影样式图案的颜色，当不指定 backcolor 时，系统取默认的黑色作为背景色。

例如，下面的代码使用 HatchBrush 类创建一个阴影画刷并填充到扇形：

```
//定义限定扇形区的矩形
Rectangle rect = new Rectangle(30, 50, 500, 500);
//创建画笔
Pen p = new Pen(Color.Tomato, 4);
float startAngel = 200.0F;
float sweepAngle = 140.0F;
//绘制饼形
g.DrawPie(p, rect, startAngel, sweepAngle);
//创建阴影画刷
HatchBrush htBrush =
  new HatchBrush(HatchStyle.DiagonalBrick, Color.Black, Color.Wheat);
//填充扇形
g.FillPie(htBrush, rect, startAngel, sweepAngle);
```

3. 纹理画刷

纹理画刷是指使用某一图像或图像的一部分来填充区域的内部，由 TextureBrush 类实现。TextureBrush 类构造函数的几种常用形式为：

```
TextureBrush(Image image);
TextureBrush(Image image, Rectangle dstRect);
TextureBrush(Image image, WrapMode wrapMode);
TextureBrush(Image image, WrapMode wrapMode, Rectangle dstRect);
```

上述构造函数中：

- image——用来指定纹理画刷的纹理图像。
- rect——代表用来构造纹理画刷的图像区域。
- wrapMode——用来指定纹理图像填充区域时的平铺方式。

例如，下面的实例使用 DrawEllipse()方法绘制一个圆形，并使用 FillEllipse()方法按指定的 imgae 图像来填充。这里填充时所使用的画刷是纹理画刷 TextureBrush，实现代码如下：

```
//定义图像文件
Bitmap myBitmap = new Bitmap(@"logo.gif");
//创建纹理画刷
TextureBrush myTextureBrush = new TextureBrush(myBitmap);
//定义填充区域的大小和位置(圆)
Rectangle rect = new Rectangle(30, 30, 500, 300);
//定义画笔
Pen p = new Pen(Color.Black, 2);
//绘制区域
g.DrawEllipse(p, 30, 30, 500, 300);
//填充区域(圆)
g.FillEllipse(myTextureBrush, rect);
```

4. 线性渐变画刷

线性渐变画刷是指在一个区域中使用两种颜色进行过渡(渐变)，渐变方向可以是水平、垂

直或对角线方向。LinearGradientBrush 类封装了线性渐变画刷的功能，常用构造函数如下：

```
LinearGradientBrush(Point point1, Point point2, Color color1, Color color2);
LinearGradientBrush(Rectangle rect, Color color1, Color color2,
  LinearGradientMode mode);
LinearGradientBrush(Rectangle rect, Color color1, Color color2, float angle);
```

在上述构造函数中：

- point1 和 point2——分别用来指定渐变矩形区域的左上角点和右下角点坐标。
- color1 和 color2——分别用来指定渐变的起始颜色和终止颜色。
- rect——用来指定渐变区域的大小和位置。
- angle——用来指定渐变方向与 X 轴的角度，取正值时表示沿顺时针方向。
- mode——用来指定渐变方向，是 LinearGradientMode 枚举类型。

LinearGradientBrush 类中的 LinearColors 属性和 Rectangle 属性分别能指定渐变的起始颜色、终止颜色和渐变区域的大小及位置。如下所示的代码使用线性渐变画刷来填充圆形：

```
//指定画布大小
Bitmap bitmap = new Bitmap(760, 240);
//创建画布
Graphics g = Graphics.FromImage(bitmap);
g.FillRectangle(Brushes.SlateGray, 1, 1, bitmap.Width - 2, bitmap.Height - 2);
Rectangle myRectangle = new Rectangle(20, 20, 160, 160);
//创建渐变画刷
LinearGradientBrush lgbRectangle1 = new LinearGradientBrush(myRectangle,
  Color.DodgerBlue, Color.Snow, LinearGradientMode.BackwardDiagonal);
LinearGradientBrush lgbRectangle2 = new LinearGradientBrush(myRectangle,
  Color.DodgerBlue, Color.Snow, LinearGradientMode.ForwardDiagonal);
LinearGradientBrush lgbRectangle3 = new LinearGradientBrush(myRectangle,
  Color.DodgerBlue, Color.Snow, LinearGradientMode.Horizontal);
LinearGradientBrush lgbRectangle4 = new LinearGradientBrush(myRectangle,
  Color.DodgerBlue, Color.Snow, LinearGradientMode.Vertical);
//填充圆
g.FillEllipse(lgbRectangle1, 20, 20, 160, 160);
g.FillEllipse(lgbRectangle2, 200, 20, 160, 160);
g.FillEllipse(lgbRectangle3, 380, 20, 160, 160);
g.FillEllipse(lgbRectangle4, 560, 20, 160, 160);
```

这里绘制了 4 个大小相同的圆，分别使用不同 mode 的填充方式，运行效果如图 10-7 所示。

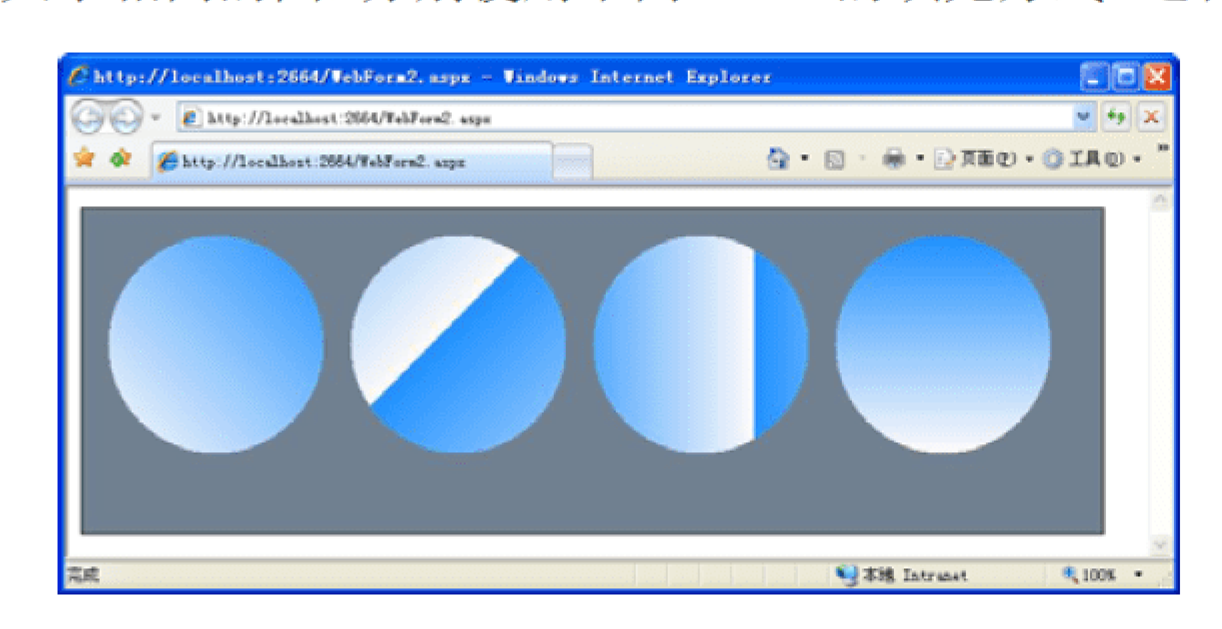


图 10-7　线性渐变

10.5.3 实例描述

在软件运行环境日益复杂的今天，安全成为大部分软件必须考虑的问题。黑客无处不在，攻击方式日益丰富，尤其是 Web 系统由于其开放性更是遇到严峻的考验，黑客事件层出不穷，造成的损失和影响也不断变大。对此，我们软件开发人员需要对此有相当的认识并采取措施抵御各种黑客攻击。

在众多的抵御手段中，采用验证码技术就是一种行之有效的办法。那么，我们平常在网页中看到的验证码图片是如何生成的呢？哈哈，这就要用到我们前面所学的图形编程技术。下面，就让我们通过一个实例来体验一下。

10.5.4 实例应用

【例 10-2】实现验证码。

验证码是由一个独立的页面来实现的，页面名称为“checkimage.aspx”。它的前台不添加任何内容，直接进入.cs 编写代码。

第一步先将绘图所需的 System.Drawing 命名空间引用到文件中。再到 Page_Load()方法中添加如下代码：

```
Random rand = new Random();
int len = rand.Next(4, 4);
char []chars =
  "0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz"
  .ToCharArray();
System.Text.StringBuilder myStr = new System.Text.StringBuilder();
for (int iCount=0; iCount<len; iCount++)
{
    myStr.Append(chars[rand.Next(chars.Length)]);
}
string text = myStr.ToString();
// 保存验证码到 Session 中以便其他模块使用
this.Session["checkcode"] = text;
```

在上述代码中，我们首先使用随机类 Random 来随机生成一个 4 位的包含数字和英文字符的文本，这就是验证码的原始文本，我们将其保存在 Session 中供以后使用。

当然，也可以使用汉字来做校验码，但是用户恐怕很难接受吧？如果遇到验证码里面的字居然不认识的时候，说不定用户会很恼火的。

然后创建一个临时图片，并据此创建一个临时的图片对象，调用 Graphics 的 MeasureString()方法获得这个字符串的显示大小，据此就可以计算出验证码图片的大小：

```
Size ImageSize = Size.Empty;
Font myFont = new Font("MS Sans Serif", 14);
// 计算验证码图片大小
using (Bitmap bmp = new Bitmap(10, 10))
{
```

```
    using (Graphics g = Graphics.FromImage(bmp))
    {
        SizeF size = g.MeasureString(text, myFont,200);
        ImageSize.Width = (int)size.Width + 8;
        ImageSize.Height = (int)size.Height + 3;
    }
}
```

然后创建一个位图对象 bmp，在此基础上创建一个图形绘制对象，调用 DrawString()方法将验证码文本绘制在这个位图上。具体代码如下：

```
// 创建验证码图片
using (Bitmap bmp = new Bitmap(ImageSize.Width, ImageSize.Height))
{
    // 绘制验证码文本
    using (Graphics g = Graphics.FromImage(bmp))
    {
        g.Clear(Color.White);
        using (StringFormat f = new StringFormat())
        {
            f.Alignment = StringAlignment.Center;
            f.LineAlignment = StringAlignment.Center;
            f.FormatFlags = StringFormatFlags.NoWrap; //不自动换行
            g.DrawString(text, myFont, Brushes.Black,
              new RectangleF(0, 0, ImageSize.Width, ImageSize.Height), f);
        }
    }
    myFont.Dispose();
}
```

验证码出现之后，还需要在图片上随机制造杂点来混淆图片内容。这些杂点的面积占图片面积的 20%，而且其位置和颜色都是随机的。这些杂点能严重干扰程序辨认验证码文本内容，但人脑可以轻松地排除这些干扰：

```
// 制造噪声杂点，面积占图片面积的 20%
int num = ImageSize.Width * ImageSize.Height * 20 / 100;
for (int iCount=0; iCount<num; iCount++)
{
    // 在随机的位置使用随机的颜色设置图片的像素
    int x = rand.Next(ImageSize.Width);
    int y = rand.Next(ImageSize.Height);
    int r = rand.Next(255);
    int g = rand.Next(255);
    int b = rand.Next(255);
    Color c = Color.FromArgb(r, g, b);
    bmp.SetPixel(x, y, c);
}
```

验证码图片生成后就可以输出到客户端了，这里采用 PNG 格式进行输出：

```
// 输出图片
System.IO.MemoryStream ms = new System.IO.MemoryStream();
bmp.Save(ms, System.Drawing.Imaging.ImageFormat.Png);
this.Response.ContentType = "image/png";
ms.WriteTo(this.Response.OutputStream);
ms.Close();
```

为了能够判断用户输入的验证码与程序生成的是否一致。这里还需要在 checkimage.aspx 中添加一个静态方法 CheckCode()来进行检测：

```
public static bool CheckCode(string text)
{
    // 检查指定的文本是否匹配验证码
    // <param name="text">要判断的文本</param>
    // <returns>是否匹配</returns>
    string txt = System.Web.HttpContext.Current
      .Session["checkcode"].ToString(); //原始验证码转为字符串
    return text==txt;
}
```

验证码页面 checkimage.aspx 完成后，我们就可以利用这个页面来实现验证码技术。接下来，建立一个系统登录页面 Login.aspx，该页面的布局如图 10-8 所示。

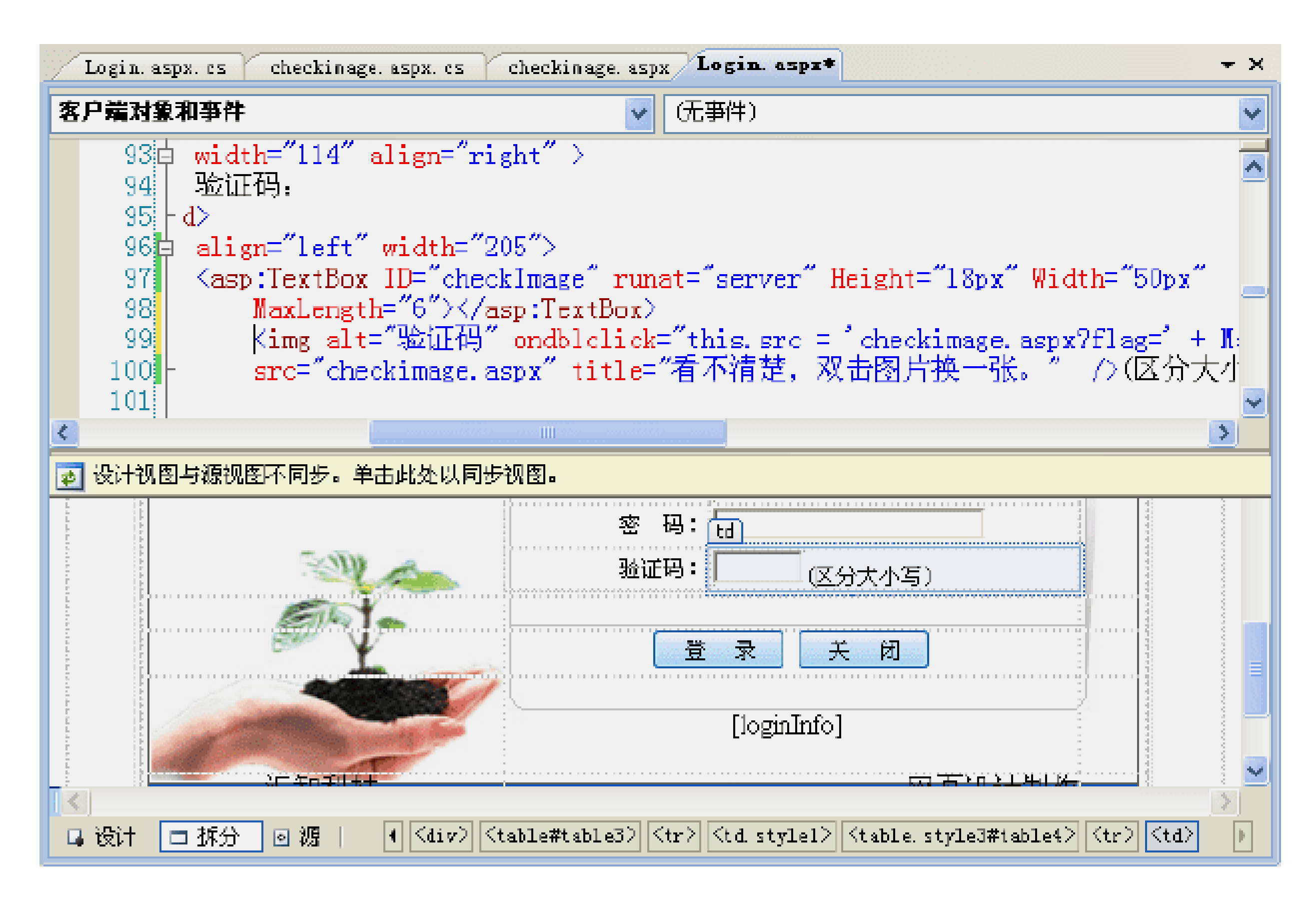


图 10-8　登录页面布局

找到页面上输入验证码的位置，添加下面的代码。包括一个 TextBox 控件和一个 img 控件，并设置相关属性值，具体代码如下：

```
<asp:TextBox ID="checkImage" runat="server" Height="18px" Width="50px"
  MaxLength="6"></asp:TextBox>
<img alt="验证码"
  ondblclick="this.src = 'checkimage.aspx?flag=' + Math.random() "
  src="checkimage.aspx" title="看不清楚，双击图片换一张。" />(区分大小写)
```

可以看到，在 ondblclick 事件处理中更新了图片来源，这里附加一个随机参数 flag，从而保证浏览器不会使用本地缓存而是每次重载最新的验证码图片。

最后，在“登录”按钮的单击事件中添加代码，判断用户名、密码以及验证码并给出提示。

具体的参考代码如下：

```
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    string UserName = this.userName.Text;
    string Password = this.userPassword.Text;
    string CheckCode = this.checkImage.Text;
    if (checkimage.CheckCode(CheckCode))    //判断验证码是否正确
    {
        loginInfo.Text = "<b>验证码输入正确。</b>";
    }
    else
    {
        loginInfo.Text = "<b>验证码输入错误。</b>";
    }
    //如果用户名、密码和验证码都正确，则登录成功
    if (UserName=="admin" && Password=="itzcn.com"
      && checkimage.CheckCode(CheckCode))
    {
        Page.ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('登录成功。');</script>");
    }
    else
    {
        Page.ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('登录失败。');</script>");
    }
}
```

10.5.5　运行结果

至此，我们的各个页面已经创建完毕，那么验证码的运行效果如何呢？还是运行一下看看效果吧。登录系统 Login.aspx 页面运行，查看效果，如图 10-9 所示。



图 10-9　Login.aspx的运行效果

在少数情况下，程序生成的验证码难以辨认，此时需要提供新的验证码。将鼠标移动到验证码上面，弹出“看不清楚，双击图片换一张”的提示，我们双击验证码，即可更新。

按提示输入内容之后单击“登录”按钮，因为验证码是区分大小写的，如图 10-10 所示为登录失败时的提示。

填写正确后提示“登录成功”，效果如图 10-11 所示。

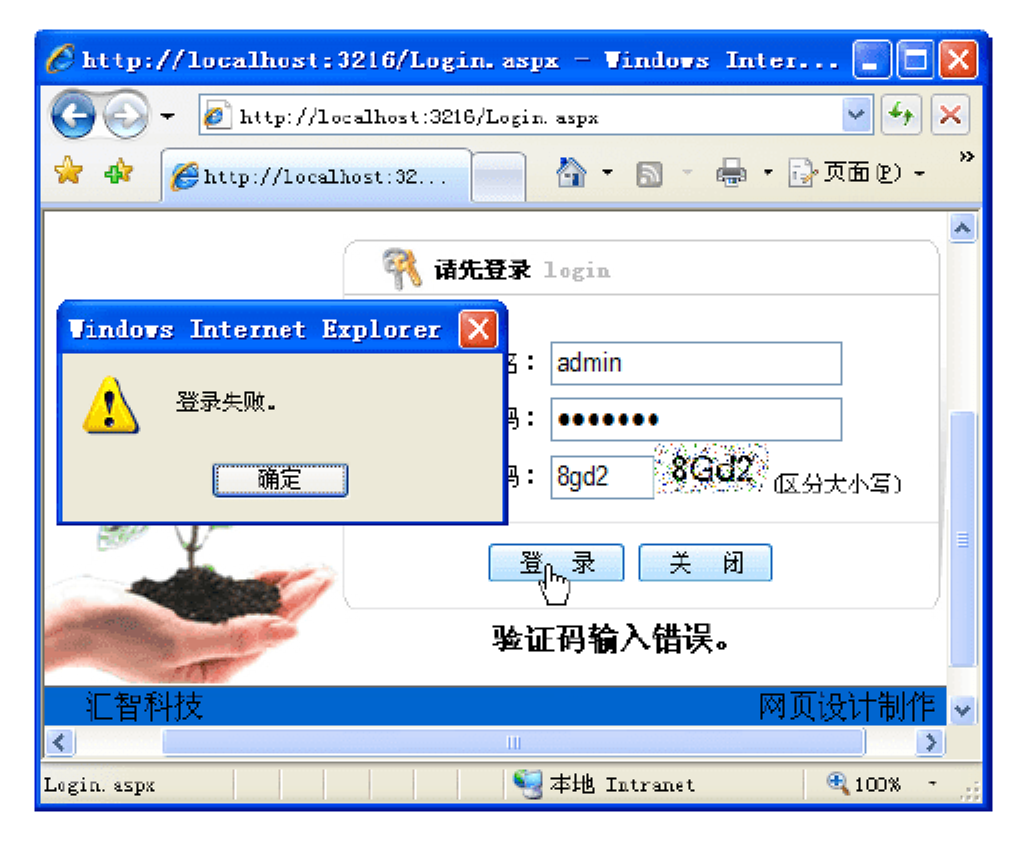


图 10-10　登录失败提示

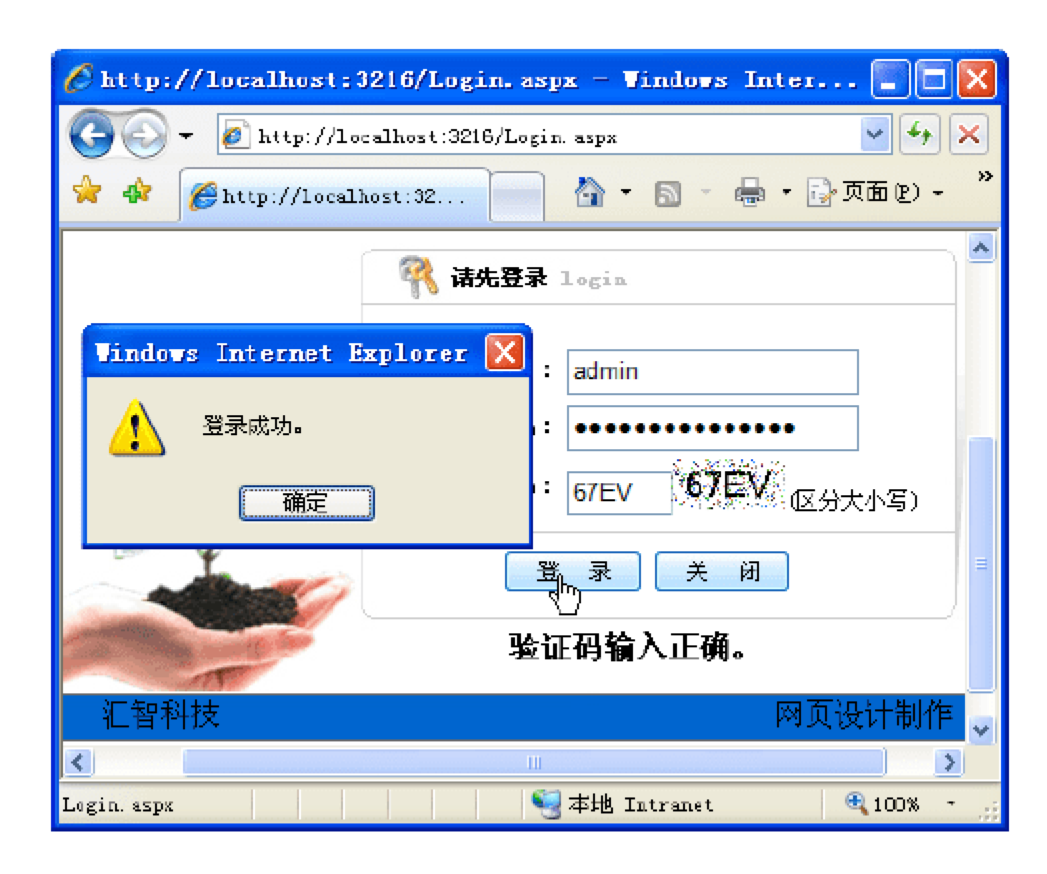


图 10-11　登录成功提示

10.5.6　实例分析



源码解析：

早听说实现验证码有一定的难度，今天经过我们层层分析，直至看到运行结果，心中颇有感慨：“原来不过如此，也不是太难啊”。

其实，在本实例中难的是一种程序设计的思想。

例如，在本实例中我们先要生成一个文本，接着计算文本的长和宽。

然后根据长和宽产生一个图片，再到图片上画文本，最后往图片上添加杂点使其难以辨认。最后再经过输出，这样一个验证码图片就诞生了。

10.6　生成缩略图

近期研究一些突破 GDI+图像操作方面的知识，记录下来。一方面算是对这几天学习知识的总结，帮助自己理解；另一方面希望对研究这方面技术的读者有所帮助；另外也希望引起其他 ASP.NET 开发者的注意，在提供缩略图时多些考虑。



视频教学：光盘/videos/10/DrawImage.avi　　　长度：5 分钟

10.6.1　基础知识——GetThumbnailImage()方法

很早以前就完成了一个通用类来处理.NET 中的图片缩放和水印等处理，说实话，从没关注过其中的画质问题。直到做了拍客，终于有人指出程序生成的图片质量不高(他们总是把图片放大到 200%、300%来查看细节，汗)。

经过多次测试，最终方案是使用.NET 的缩略图方法 GetThumbnailImage()来创建。

GetThumbnailImage()是 Image 类的实例方法，其完整声明格式如下：

```
public Image GetThumbnailImage(
    int thumbWidth,
    int thumbHeight,
    Image..::..GetThumbnailImageAbort callback,
    IntPtr callbackData
)
```

可以看到，该方法执行后会返回表示缩略图的 Image 实例。各个参数的含义如下。

- thumbWidth：请求的缩略图的宽度(以像素为单位)。
- thumbHeight：请求的缩略图的高度(以像素为单位)。
- callback：一个 Image.GetThumbnailImageAbort 委托。必须创建一个委托并将对此委托的引用作为 callback 参数传递，但不使用此委托。
- callbackData：System.IntPtr 类型，值必须为 Zero。

例如，下面的代码演示如何创建并显示一个缩略图像：

```
public bool ThumbnailCallback()     //创建委托
{
   return true;
}
public void Example_GetThumb(PaintEventArgs e)
{
   Image.GetThumbnailImageAbort myCallback =
     new Image.GetThumbnailImageAbort(ThumbnailCallback);
   Bitmap myBitmap = new Bitmap("Climber.jpg");
   Image myThumbnail =
     myBitmap.GetThumbnailImage(40, 40, myCallback, IntPtr.Zero);
   Graphics.DrawImage(myThumbnail, 150, 75);
}
```

在使用 GetThumbnailImage()方法时要注意：如果 Image 包含一个嵌入式缩略图像，则此方法会检索嵌入式缩略图，并将其缩放为所需大小。而对于不包含嵌入式缩略图像的 Image，此方法会通过缩放主图像创建一个缩略图像。

10.6.2　实例描述

进过专业图片网站的读者都会发现，我们在浏览图片的时候网页中会先出现一系列小图。

然后如果想欣赏哪幅图片的话，就直接点击查看原图，网页中便会显示一张原始大小的图片。那么这张小图是如何形成的呢，会不会直接就是将大图控制尺寸显示出来的呢？答案不用说就是否定的，因为那样的话，我们打开网页的速度就会非常慢。

那究竟是如何生成的呢？下面就让我们带着这个疑问进入本节的实例——生成缩略图。

10.6.3 实例应用

【例 10-3】生成缩略图。

打开开发工具 VS2008，在页面中制作生成缩略图的布局。需要 FileUpload 控件用于上传图片文件；两个 TextBox 文本框用于设置和显示图片大小；两个 Button 按钮用于完成缩放和显示图片的功能，效果如图 10-12 所示。

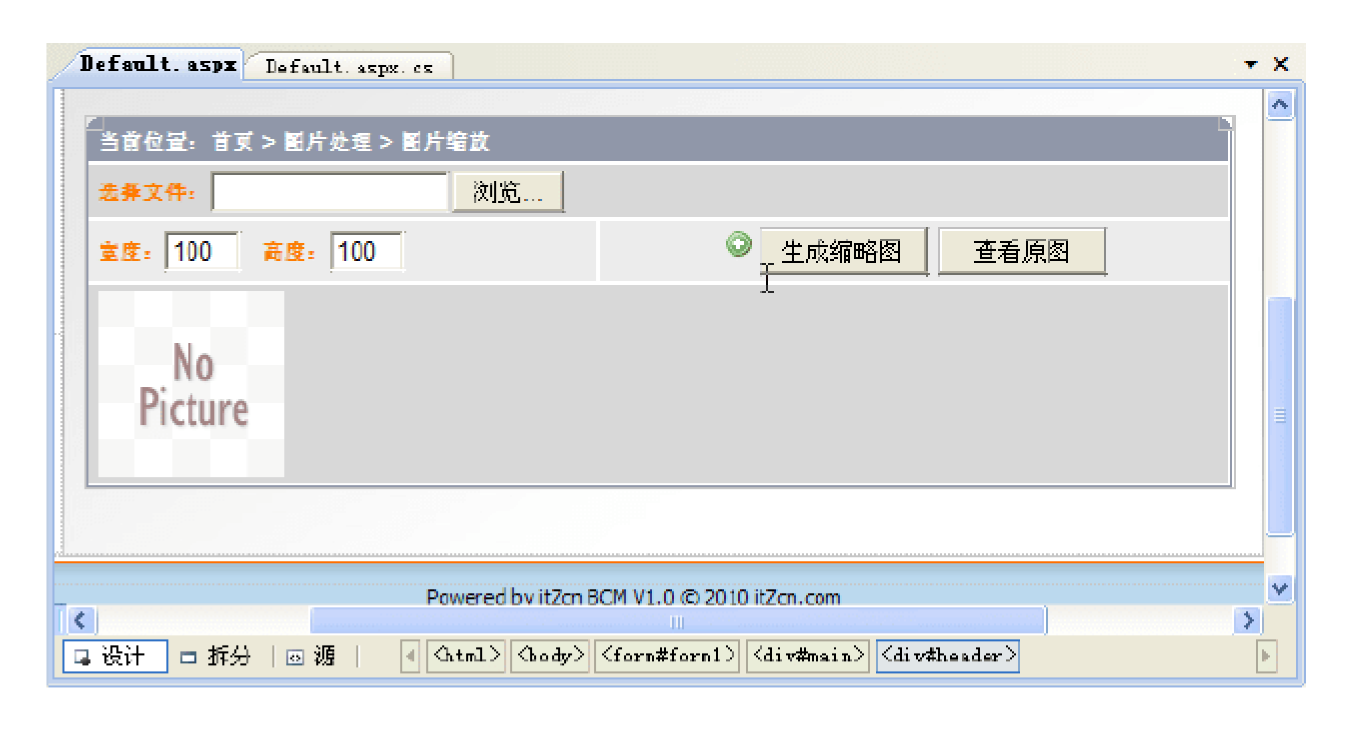


图 10-12　网页局部布局

对于布局就不再过多介绍。下面，我们双击“生成缩略图”按钮进入后台代码，在 Button1_Click()事件中添加如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    int width, height;
    width = Int32.Parse(txtWidth.Text);         //缩略图宽度
    height = Int32.Parse(txtHeight.Text);       //缩略图高度
    if (!(uploadFile.PostedFile.ContentLength > 0))
    {
        Page.ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('没有选择文件！');</script>");
    }
    else
    {
        string path = Server.MapPath("./Uploads/"
          + uploadFile.PostedFile.FileName.Substring(
          uploadFile.PostedFile.FileName.LastIndexOf('\\') + 1));
        if (File.Exists(path))
        {
            Page.ClientScript.RegisterStartupScript(
              this.GetType(), "", "<script>alert('已经有同名文件！');</script>");
```

```
        }
        else
        {
            uploadFile.PostedFile.SaveAs(path);       //上传原始图
            GetSmallImage(width, height);             //生成缩略图
        }
    }
}
```

从上述代码中我们看到，在将原始图片上传之后，接着调用了 GetSmallImage()方法。该方法主要用于完成图片缩放的功能，代码如下所示：

```
public void GetSmallImage(int width, int height)
{
    string file = "Uploads/" + uploadFile.PostedFile.FileName.Substring(
      uploadFile.PostedFile.FileName.LastIndexOf('\\')
      + 1);  //设置原始图片保存路径
    oldfile = file;
    string newfile = "Uploads/small_" + uploadFile.PostedFile.FileName
      .Substring(uploadFile.PostedFile.FileName.LastIndexOf('\\')
      + 1);  //设置缩略图片保存路径
    System.Drawing.Image oldimage =
      System.Drawing.Image.FromFile(Server.MapPath(file));
    System.Drawing.Image thumbnailImage = oldimage.GetThumbnailImage(width,
      height, new System.Drawing.Image
      .GetThumbnailImageAbort(ThumbnailCallback), IntPtr.Zero); //生成缩略图
    Response.Clear();
    Bitmap output = new Bitmap(thumbnailImage);
    Graphics g = Graphics.FromImage(output);
    output.Save(Server.MapPath(newfile),
      System.Drawing.Imaging.ImageFormat.Jpeg);
    ImgPreview.Visible = true;                                //显示缩略图
    ImgPreview.ImageUrl = newfile;
}
bool ThumbnailCallback()
{
    return true;
}
```

生成缩略图功能的代码我们编写完成了。下面，再来为“查看原图”按钮的单击事件添加代码，用来完成查看原图以及原图大小的功能。代码如下：

```
protected static  string oldfile = null;  //全局静态变量，用于存放带路径的原图名称
protected void Button2_Click(object sender, EventArgs e)
{
    ImgPreview.ImageUrl = oldfile;          //显示原图
    string path = MapPath(oldfile);
    System.Drawing.Bitmap oldpic = new System.Drawing.Bitmap(path);
    txtWidth.Text = oldpic.Width.ToString();    //显示原图宽度
    txtHeight.Text = oldpic.Height.ToString();  //显示原图高度
```

```
    oldpic.Dispose();
}
```

10.6.4　运行结果

至此，后台代码编写完成。还是运行一下网页来看看效果如何吧。在打开的网页中，我们单击“浏览”按钮，选择一张图片，然后指定要缩放后的图片宽高，最后单击“生成缩略图”按钮，缩略图便显示在网页中，如图 10-13 所示。



图 10-13　生成缩略图的效果

然后我们单击“查看原图”按钮，原图便显示在网页中，在这里可以看到原图的大小，如图 10-14 所示。



图 10-14　点击“查看原图”的效果

10.6.5　实例分析



源码解析：

在本实例中，我们用到了一个 GetThumbnailImage()方法来生成缩略图，并通过两个参数来设置生成缩略图的大小。这个方法看似灵活，但是有一个弊端，就是生成的缩略图可能会变形。

其实这个问题也很好解决，在实际应用中，我们通常通过限定缩略图的大小来进行等比缩放。具体做法如下：定义允许图片宽度和高度，当原始图片的宽高比大于等于限定图片的宽高比时，计算宽度比，高度按这个比例计算；反之，计算高度比，宽度按这个比例计算。

10.7　给原始图加水印

不知道大家有没有遇到这种情况：自己辛辛苦苦拍摄的佳作传到网络与网友分享。不久就在其他论坛上看到自己的照片，而此时的作者已不再是自己……

对于这种情况，我们除了对那些窃取别人劳动果实的人予以痛斥外，还有什么办法吗？有的，那就是为你的图片加上水印！加了水印的图片让窃图之人望“图”生畏。

下面就请出我们的专业老师“ASP.NET”出场，详细介绍制作步骤。



视频教学：光盘/videos/10/DrawString.avi　　　　**长度：**9 分钟

10.7.1　基础知识——绘制文本

在绘制文本时需要指定文本所使用的字体。字体是文字显示和打印的外在形式，包括文字的字样、风格和尺寸 3 个主要属性。对于汉字，字样是指宋体、楷体等；字体风格是指字体的粗细、是否倾斜等特点；字体尺寸是指初号、小初、二号等。在 GDI+中，Drawing.Font 类用来代表字体，要创建字体，可使用下列构造函数：

```
Font(FontFamily family, float emSize);
Font(String familyName, float emSize);
Font(FontFamily family, float emSize, FontStyle style);
Font(FontFamily family, float emSize, GraphicsUnit unit);
Font(String familyName, float emSize, FontStyle style);
Font(String familyName, float emSize, GraphicsUnit unit);
```

例如，下面的代码可以用来创建一种“宋体、5 磅、斜粗体和带下划线”的字体：

```
Drawing.Font font = new Drawing.Font("宋体", 5,
    (FontStyle)(((FontStyle.Bold|FontStyle.Italic)|FontStyle.Underline)),
    GraphicsUnit.Point);
```

GDI+为绘制文本提供了 Graphics 类的 DrawString()方法，该方法有如下几种重载形式：

```
DrawString(String str,Font font, Brush brush, PointF point);
DrawString(String str, Font font,Brush brush, PointF point,
  RectangleF layoutRectangle);
DrawString(String str, Font font, Brush brush, PointF point,
  StringFormat format);
DrawString(String str, Font font, Brush brush, PointF point,
  RectangleF layoutRectangle, StringFormat format);
DrawString(String str, Font font, Brush brush, float x, float y);
```

```
DrawString(String str, Font font, Brush brush, float x, float y,
  StringFormat format);
```

在上述格式中：

- str——用来指定要绘制的字符串。
- font——用来指定字体，是 Font 类型。
- brush——用来指定画刷，可以是单色、阴影、纹理或者渐变画刷等。
- layoutRectangle——是一个矩形区域，用来限定输出文本的范围。
- format——用来指定输出文本的格式化属性。
- point——该参数和(x，y)参数用来指定文本输出的起点。

10.7.2　基础知识——绘制图像

GDI+支持大多数流行的图像文件格式，它的 Image 类封装了对 BMP、GIF、JPEG、PNG、TIFF、WMF 和 EMF 格式图像文件进行加载、格式转换以及简单处理的功能。从 Image 类继承而来的 Bitmap 类封装了 Windows 位图操作的常用功能。

在绘制图像时，首先使用 Image 类或 Bitmap 类把图像加载到内存，从而构造一个 Image 类或 Bitmap 类的对象。例如：

```
Image imgCurrent;
imgCurrent = Image.FromFile("sky.bmp");
```

然后使用 Graphics.DrawImage 方法在指定位置处绘制全部或部分图像。该方法常用的重载形式如下：

```
void DrawImage(Image image, int x, int y);
void DrawImage(Image image, Point point);
void DrawImage(Image image, Point []destPoints);
void DrawImage(Image image, Rectangle rect);
void DrawImage(Image image, int x, int y, Rectangle srcRect,
  GraphicsUnit srcUnit);
```

在上述格式中：

- x,y 或 point 参数——用来指定要绘制图像左上角点的位置。
- destPoints 数组——用来指定一个多边形的顶点。
- rect 参数——用来指定被图像填充的矩形区域。
- srcRect 参数——表示要绘制的源图像的位置和大小。
- srcUnit 参数——表示所使用的单位，默认单位是 GraphicsUnit.Pixel(像素)。

例如：

```
Image imgCurrent;
Point pntImage = new Point(0, 0);
Rectangle recImage = new Rectangle(50, 50, 90, 180);
imgCurrent = Image.FromFile("green.bmp");
Graphics.DrawImage(imgCurrent, pntImage);
Graphics.DrawImage(imgCurrent, recImage);
```

以上代码中 pntImage 指定绘制图像时的起点位置；recImage 指定将被图像填充的矩形区域。Image.FromFile 方法从文件 green.bmp 创建了一个 Image 对象实例，然后将其存入 recImage 中。第一次使用 Graphics.DrawImage 方法绘制出的图像是原始大小的图像，第二次绘制的图像被限制在 recImage 所定义的矩形区域内。

10.7.3　实例描述

如今，越来越多的网民都喜欢将自己数码照片传到网上，与亲戚、朋友分享；大部分企业也是借助互联网，发布自己的产品展示图片，来推广自己的产品……但是，在互联网上传播、下载图片是非常容易的，这就造成很多图片在没有经过著作权人同意的情况下被到处流传，甚至构成侵权。为了使自己的图片不被滥用，同时又起到保护所有权的目的，在图片上传的时候给图片加上一些水印信息，确实是一个非常奏效的方法。

下面就让我们通过实例来体验一下，如何用 ASP.NET 在上传过程中为图片加上水印。

10.7.4　实例应用

【例 10-4】给原始图加水印。

首先我们需要让用户选择一个要添加水印的图片，这可以用 FileUpload 控件来完成。再添加两个 Button 控件，分别写上“加文字水印”和“加图片水印”，供用户选择水印的类型。接着在下方添加一个 Image 控件，为 ImageUrl 属性指定一张图片的路径，该图片在运行后作为默认显示。再调整各个主要控件的位置，最终类似图 10-15 所示的布局。

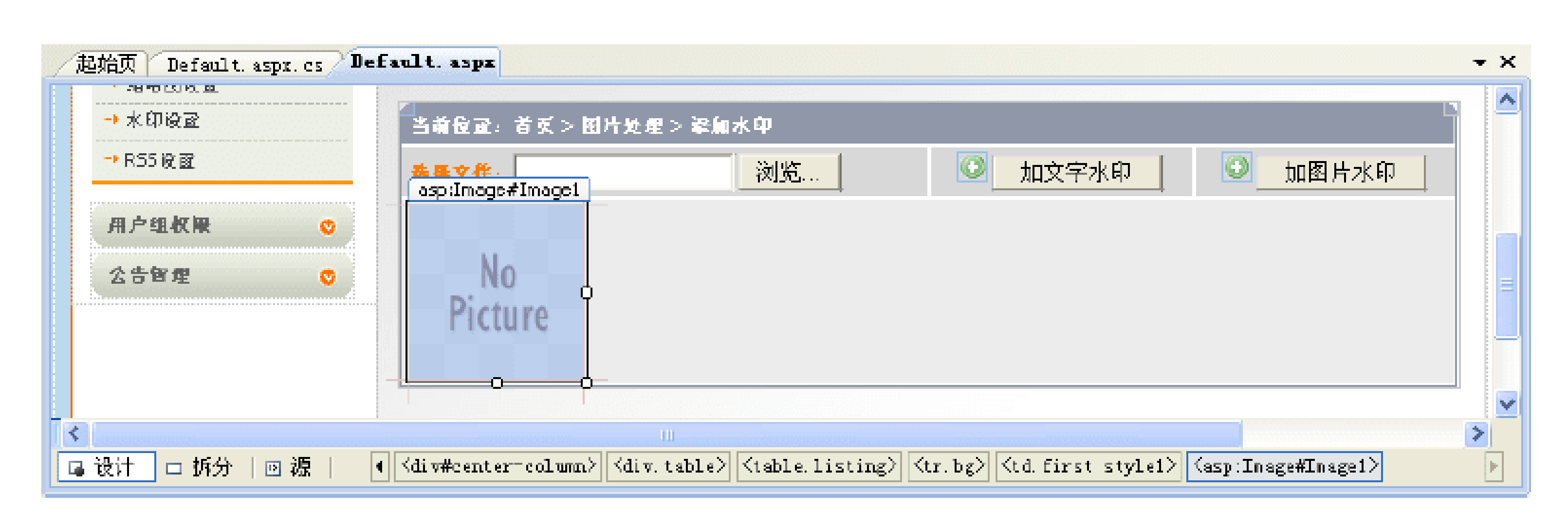


图 10-15　添加布局

现在，前台的布局设计就算完成了。下面就来编写实现上传并添加水印的代码，首先从“加文字水印”按钮开始。在按钮的单击事件中添加如下代码：

```
protected void btAddWeater1_Click(object sender, EventArgs e)
{
    if (fuAddCate.PostedFile.FileName.Trim() != "")
    {
        //上传文件
        string extension =
          Path.GetExtension(fuAddCate.PostedFile.FileName).ToLower();
        string fileName = DateTime.Now.ToString("yyyyMMddhhmmss");
        string path = Server.MapPath("~/ImgUpload/") + fileName + extension;
        fuAddCate.PostedFile.SaveAs(path);
```

```
        //加文字水印
        System.Drawing.Image image = System.Drawing.Image.FromFile(path);
        Graphics g = Graphics.FromImage(image);
        g.DrawImage(image, 0, 0, image.Width, image.Height);
        Font f = new Font("宋体", 20);
        Brush b = new SolidBrush(Color.Red);
        String addText = "汇智科技";
        g.DrawString(addText, f, b, 20, 20);
        g.Dispose();
        //保存，并删除原始照片
        string newPath = Server.MapPath("~/ImgUpload/")
          + fileName + "_wm" + extension;
        image.Save(newPath);
        image.Dispose();
        if (File.Exists(path))
        {
            File.Delete(path);
        }
        if (File.Exists(newPath))
        {
            Image1.ImageUrl = "~/ImgUpload/" + fileName + "_wm" + extension;
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
              "<script>alert('文字水印添加成功！');</script>");
        }
        else
        {
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
              "<script>alert('提示：图片上传失败！');</script>");
        }
    }
    else
    {
        Page.ClientScript.RegisterStartupScript(this.GetType(), "",
          "<script>alert('提示：您还没有选择图片！');</script>");
    }
}
```

打开“加图片水印”按钮的 Click 事件，在此编写添加图片水印的代码。这一步我们还需要准备一张图片“logo.jpg”，再放到站点的根文件夹下作为水印 Logo 图片。该事件的部分代码如下(其他可参考文字水印的代码):

```
//加图片水印
System.Drawing.Image image = System.Drawing.Image.FromFile(path);
System.Drawing.Image copyImage =
  System.Drawing.Image.FromFile(Server.MapPath("~/logo.jpg"));
Graphics g = Graphics.FromImage(image);
//在左上角绘制 logo
g.DrawImage(copyImage, 0, 0, copyImage.Width, copyImage.Height);
g.Dispose();
```

如果想在右下角出现水印 Logo，只需调整 g.DrawImage()的参数列表即可，代码如下：

```
g.DrawImage(copyImage,
  new Rectangle(image.Width - copyImage.Width, image.Height - copyImage.Height,
  copyImage.Width, copyImage.Height),
  0, 0, copyImage.Width, copyImage.Height, GraphicsUnit.Pixel);
```

这样功能页面就设计完毕。

10.7.5　运行结果

下面在浏览器执行，查看一下图片上传后的效果吧。单击“浏览”按钮，从弹出的对话框中选择一个文件，再单击“加文字水印”按钮。然后会提示添加成功，并在下方显示加水印后的图片，效果如图 10-16 所示。

再次运行，选择一个文件后，单击“加图片水印”按钮，此时的效果如图 10-17 所示。

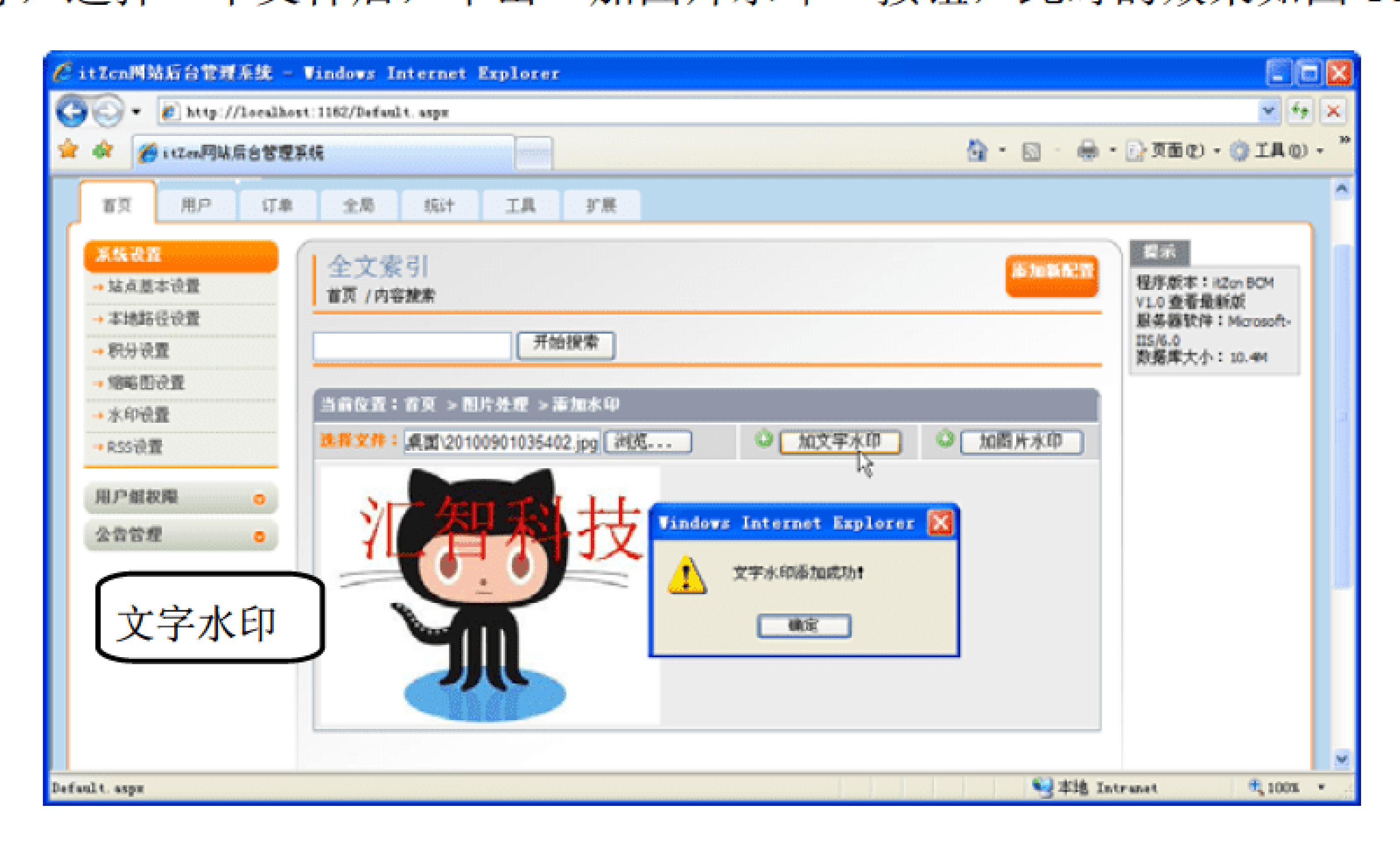


图 10-16　文字水印的效果

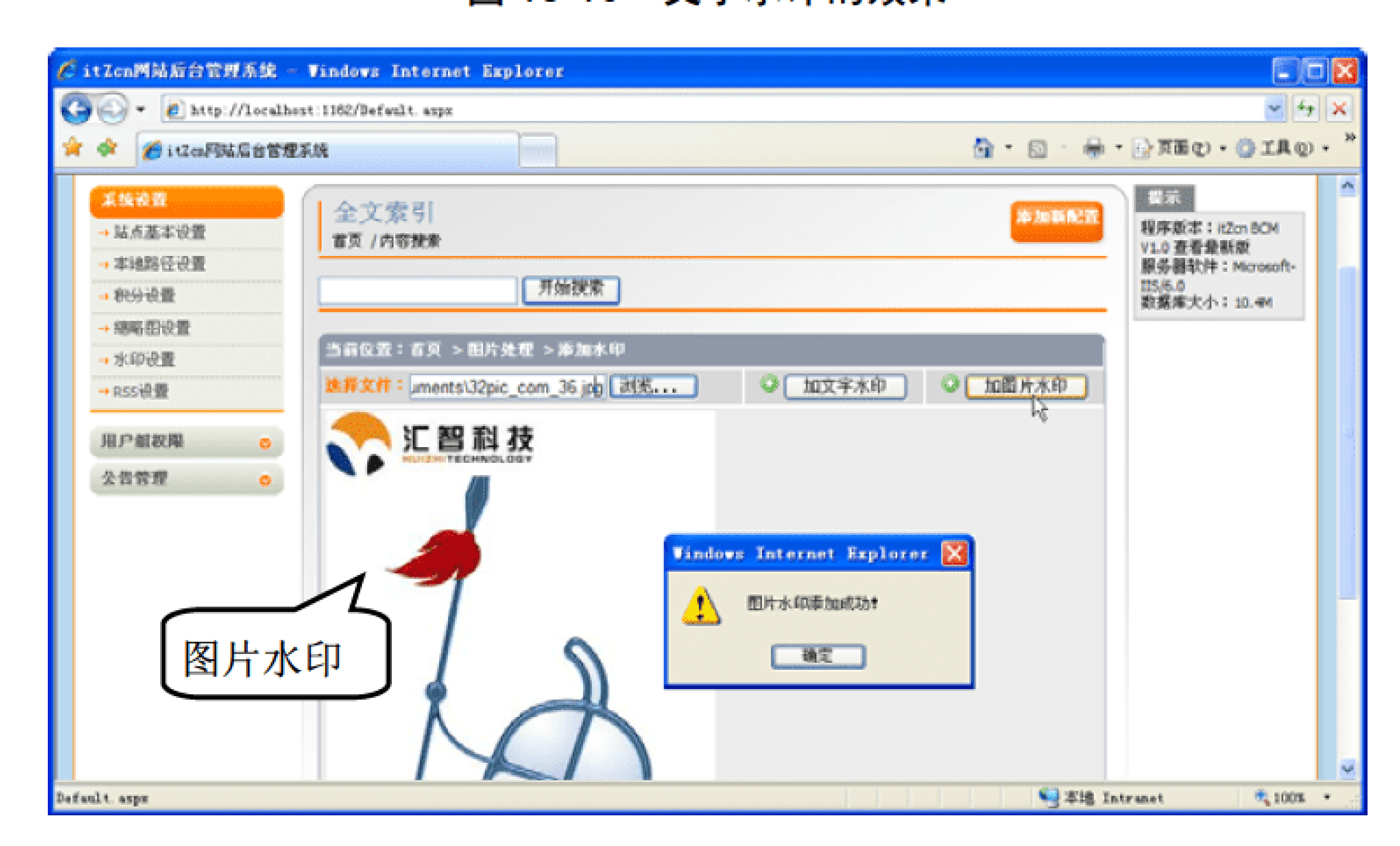


图 10-17　图片水印的效果

10.7.6 实例分析



源码解析：

通过以上分析，我们发现给图片加水印只需用 DrawString()和 DrawImage()两个方法便可实现。但是，这两个方法非常灵活，而且具有的重载形式有很多种。读者只有熟练掌握这两个方法中各个参数的意义，才能根据自身的需要进行设置。

10.8 绘制饼状图

饼状图将所有统计数据的总和看成一个圆，而把其中每一部分统计数据看成是组成圆的扇形。这些扇形大小是根据每一部分数据的大小决定的。



视频教学： 光盘/videos/10/draw.avi　　　　长度：6 分钟

10.8.1 基础知识——DrawPie()和FillPie()方法

饼状图看似一个圆饼按照一定的数值比例分割而成，实际上在具体实现时是由许多扇形组合而成。在绘制扇形时用到最多的就是 Graphics 类中的 DrawPie 方法和 FillPie 方法，分别用于绘制扇形和填充扇形。

DrawPie()方法具有多种定义方式，如下所示。

- DrawPie(Pen pen, Rectangle rect, float startAngle, float sweepAngle)：rect 定义该扇形所属的椭圆边框，startAngle 指定从 x 轴到扇形的第一条边沿顺时针方向度量的角(以度为单位)，sweepAngle 是从 startAngle 参数到扇形的第二条边沿顺时针方向度量的角。
- DrawPie(Pen pen, RectangleF rectf, float startAngle, float sweepAngle)：该方法坐标和角度值可以为 Float 型，用法与上述方法相同。
- DrawPie(Pen pen, int x, int y, int width, int height, int startAngle, int sweepAngle)：该方法中 x、y、width 和 height 用来指定所属椭圆边框的起始点坐标及其宽度和高度。
- DrawPie(Pen pen, float x, float y, float width, float height, float startAngle, float sweepAngle)：该方法定义其参数时可以为 float 型，其用法与上一个方法相同。使用 FillPie()方法时，除了第一个参数为 Brush 外，其他参数和使用方法均与 DrawPie()方法相同。

10.8.2 实例描述

饼状图对大家来说也许并不陌生，它可以很直观地反映数据统计的情况。

我们在日常学习、工作和生活中会经常见到一些统计数据以饼状图的形式表现出来，如图

10-18 所示。那么，这些饼状图是如何绘制出来的呢，以及在网页中又是如何动态地显现出来的呢？

下面就让我们带着这些疑问进入本节的实例——绘制饼状图。

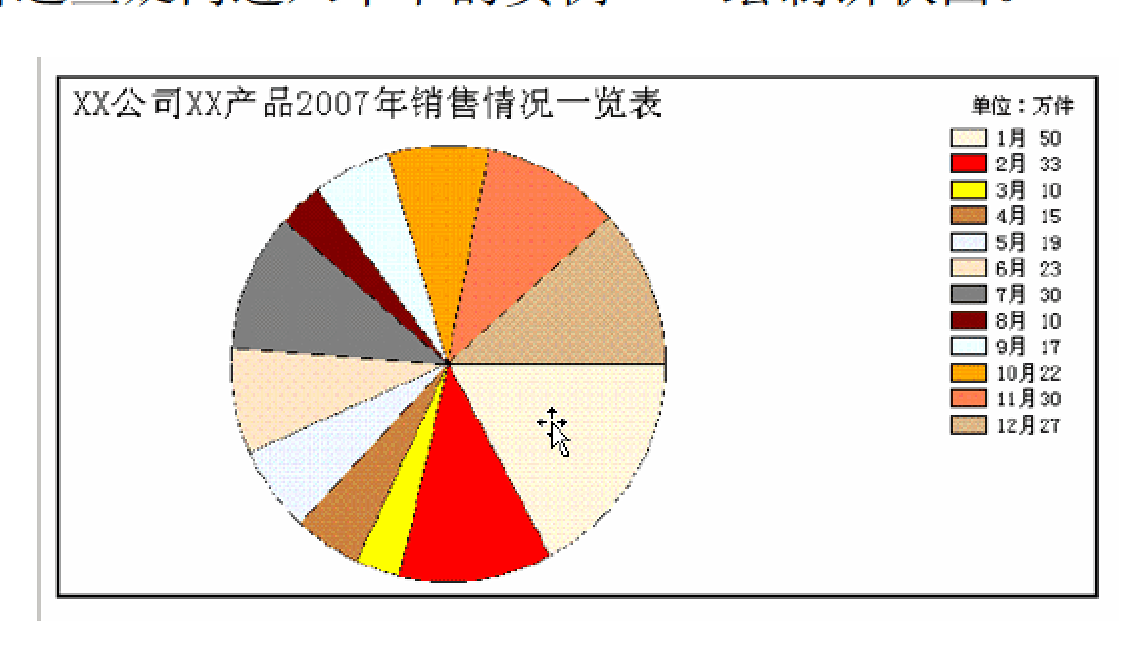


图 10-18　饼形图示例

10.8.3　实例应用

【例 10-5】绘制饼状图。

在 VS2008 中新建一个 PieChart.aspx 文件，用于绘制饼状图。该页面中我们不添加任何内容，直接打开 PieChart.aspx.cs 文件，在 Page_Load()事件中编写实现代码。

画图的第一步，我们先创建了一块 600×360 大小的画布，然后在画布上绘制图表的标题、并声明变量。具体代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    //设置请求格式为 JPG
    Response.ContentType = "image/Jpeg";
    //指定画布大小
    Bitmap bitmap = new Bitmap(600, 360);
    //创建画布
    Graphics g = Graphics.FromImage(bitmap);
    g.FillRectangle(Brushes.White, 1, 1, bitmap.Width - 2, bitmap.Height - 2);
    //绘制标题
    g.DrawString("汇智科技网站访问量统计", new Font("宋体", 20),
      new SolidBrush(Color.Black), new Point(100, 30));
    int sum = 0;
    //准备数据
    int [,]strMonthInfo = new int[,] { { 01, 30000 }, { 02, 25000 }, { 03, 35000 },
      { 04, 30000 }, { 05, 28000 }, { 06, 40000 } };
    for (int i=0; i<strMonthInfo.Length/2; i++)
    {
        sum += strMonthInfo[i, 1];
    }
    float startAngle = 0.0f;              //起始角度
    float sweepAngle = 0.0f;
}
```

上述代码中，数组变量 strMonthInfo 用于存放月份和访问量相关的数据。在通常情况下，这部分数据是要从数据库中获得的。最后的循环用于获得各月份访问量的总和，以便后面计算各部分所占比例时使用。

由于在绘制饼形图时各块扇形区域需要填充不同的颜色。所以，干脆我们用一个 for 循环，用于指定从 1 月份到 6 月份的饼形图的颜色。

代码如下：

```
//定义组成饼形图的各个扇形的颜色
Pen p = null;
SolidBrush sb = null;

for (int i=0; i <strMonthInfo.Length/2; i++)
{
   if (i == 0)
   {
      p = new Pen(Color.Red);
      sb = new SolidBrush(Color.Red);
   }
   else if (i == 1)
   {
      p = new Pen(Color.Blue);
      sb = new SolidBrush(Color.Blue);
   }
   else if (i == 2)
   {
      p = new Pen(Color.Green);
      sb = new SolidBrush(Color.Green);
   }
   else if (i == 3)
   {
      p = new Pen(Color.CornflowerBlue);
      sb = new SolidBrush(Color.CornflowerBlue);
   }
   else if (i == 4)
   {
      p = new Pen(Color.Indigo);
      sb = new SolidBrush(Color.Indigo);
   }
   else
   {
      p = new Pen(Color.Violet);
      sb = new SolidBrush(Color.Violet);
   }
}
```

接下来，我们根据每月的访问量计算出每个扇形区域所占用的角度，然后根据上面定义的颜色来绘制扇形，并进行填充，从而组成饼状图。

Web程序员成功之路

继续在 for 循环中添加如下代码：

```
//绘制饼形

float flMonthNumber = float.Parse(strMonthInfo[i, 1].ToString());
sweepAngle = 360 * (flMonthNumber / sum);
Rectangle rtl = new Rectangle(50, 100, 200, 200);
g.DrawPie(p, rtl, startAngle, sweepAngle);
g.FillPie(sb, rtl, startAngle, sweepAngle);
startAngle += sweepAngle;
```

饼形图绘制完毕后用户并不知道各部分颜色的含义，所以还要绘制说明。这里，我们以绘制不同颜色文本的方式向用户说明饼形图各部分的含义。

具体代码如下：

```
//绘制饼形图的说明

string strMonth = strMonthInfo[i, 0].ToString() + "月份所占比例："
  + (int)((flYearNumber / sum) * 100) + "%";

Font font = new Font("宋体", 12, FontStyle.Bold);

Point point = new Point(320, (110 + i * 30));
g.DrawString(strMonth, font, sb, point);
```

最后，我们创建一个输出流，将图片输出，并释放资源。该部分代码放在 for 循环外，具体如下：

```
//创建一个输出流
System.IO.MemoryStream ms = new System.IO.MemoryStream();

//将画布的内容输出到流
bitmap.Save(ms, System.Drawing.Imaging.ImageFormat.Jpeg);

//输出流
Response.ClearContent();
Response.BinaryWrite(ms.ToArray());

//释放资源
bitmap.Dispose();
g.Dispose();
Response.End();
```

经过以上代码的编写，我们就完成了一个饼状图的绘制。

接下来，我们新建一个 Default.aspx 网页，并在合适的位置通过如下的代码将饼状图引入页面中：

```
<img alt="饼形图" src="PieChart.aspx"/>
```

10.8.4　运行结果

赶快保存文件，运行一下网页吧，效果如图 10-19 所示。



图 10-19　绘制饼状图的效果

10.8.5　实例分析


源码解析：

从整体上来看，在绘制扇形时用到最多的就是 Graphics 类中的 DrawPie 方法和 FillPie 方法，分别用于绘制扇形和填充扇形。这两个方法在使用的过程中需要注意的是参数 startAngle 与 sweepAngle。

startAngle 指定从 X 轴到扇形的第一条边沿顺时针方向度量的角(以度为单位)，sweepAngle 是从 startAngle 参数到扇形的第二条边沿顺时针方向度量的角。

在本实例中，我们用“startAngle += sweepAngle;”很方便地计算出绘制下一个扇形的起始角度。

10.9　绘制柱状图

柱状图和饼状图是两种常见图表的形式。饼状图是以圆形为基础，分割成不同的扇形，并填充不同的颜色来表现数据；而柱状图是以不同颜色的矩形条来表现数据的内容，如图 10-20 所示。

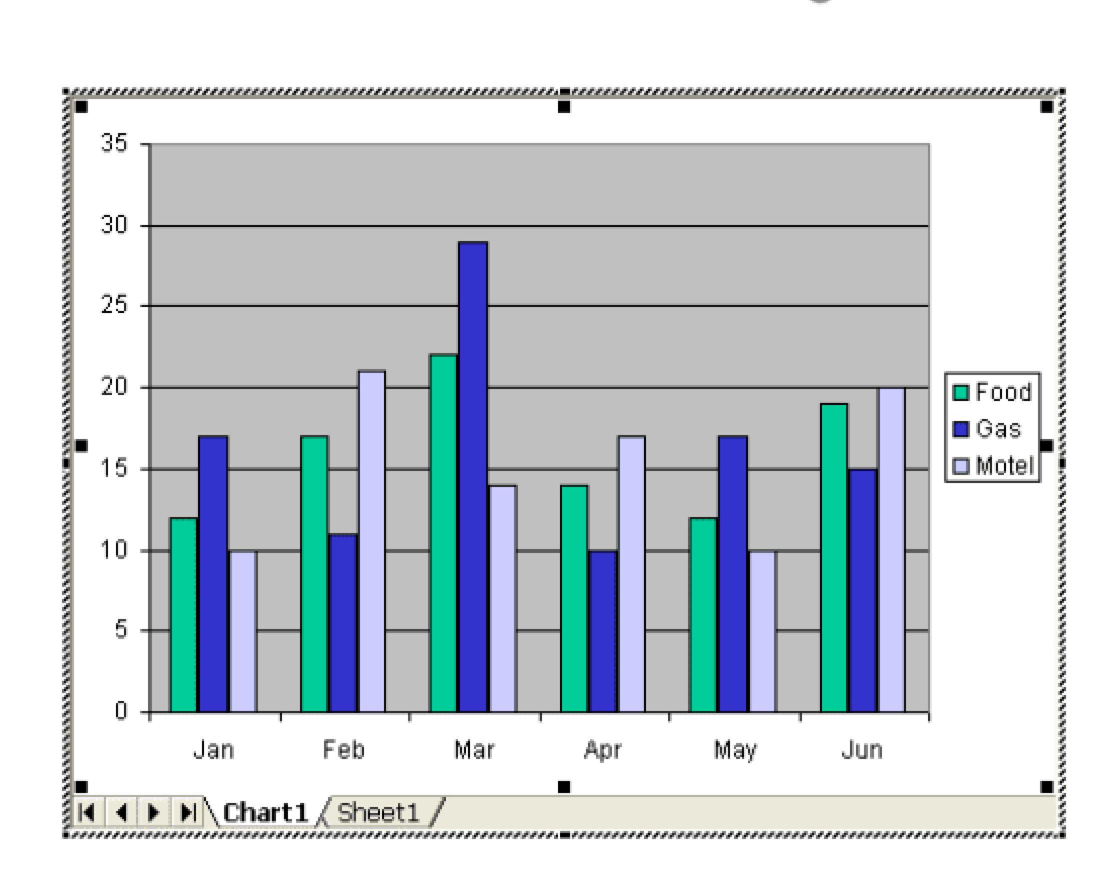


图 10-20　柱状图示例

 视频教学：光盘/videos/10/draw.avi　　　　长度：6 分钟

10.9.1　基础知识——DrawRectangle()和FillRectangle()方法

柱形图表是根据矩形的长短来表示数据。而在绘制矩形时用到最多的就是 Graphics 类中的 DrawRectangle()方法和 FillRectangle()方法，分别用于绘制矩形和填充矩形。

这两个方法的语法格式如下：

```
void DrawRectangle(Pen pen, Rectangle rect)
void DrawRectangle(Pen pen, int x, int y, int width, int height)

void FillRectangle(Brush brush, Rectangle rect)
void FillRectangle(Brush brush, int x, int y, int width, int height)
```

前面两个 Draw 方法用来绘制轮廓，后面两个 Fill 方法用来填充轮廓的内部区域。其中 pen 是一个 Pen 类结构，它确定曲线的颜色、宽度和样式；brush 是一个确定填充特性的 Brush 对象；rect 是一个 Rectangle 结构，它定义矩形的边界。x 定义矩形的边框的左上角的 X 坐标；y 定义矩形的边框的左上角的 Y 坐标；width 定义矩形的边框的宽度；height 定义矩形的边框的高度。

例如，下面这段代码演示了如何为矩形区域绘制内边框和外边框：

```
Rectangle rect = new Rectangle(10, 10, 40, 30);
g.FillRectangle(Brushes.LightBlue, rect);

Rectangle innerBounds =
  new Rectangle(rect.Left, rect.Top, rect.Width-1, rect.Height-1);

Rectangle outerBounds =
  new Rectangle(rect.Left-1, rect.Top-1, rect.Width+1, rect.Height+1);

g.DrawRectangle(Pens.Brown, innerBounds);
g.DrawRectangle(Pens.Blue, outerBounds);
```

10.9.2 实例描述

上一节中我们讲到了饼状图的绘制方法，对绘制图表的方法有了一定的理解，下面就让我们通过绘制柱状图实例，进一步学习一下图表的绘制方法(这只是作者的解决方法)。

10.9.3 实例应用

【例 10-6】绘制柱状图。

根据惯例，我们先新建一个 ColumnChart.aspx 文件，用于绘制柱状图。接着直接打开 ColumnChart.aspx.cs 文件，在 Page_Load()事件中开始编写代码。

我们先创建了一块 600×360 大小的画布，然后在画布上绘制图表的标题、X 轴、Y 轴及相关文本。具体代码如下：

```
protected void Page Load(object sender, EventArgs e)
{
    //设置请求格式为 jpg
    Response.ContentType = "image/Jpeg";
    //指定画布大小
    Bitmap bitmap = new Bitmap(600, 360);
    //创建画布
    Graphics g = Graphics.FromImage(bitmap);
    g.FillRectangle(Brushes.White, 1, 1, bitmap.Width - 2, bitmap.Height - 2);
    //绘制标题
    g.DrawString("汇智科技网站访问量统计", new Font("宋体", 20),
      new SolidBrush(Color.Black), new Point(150, 30));
    //绘制 X 轴
    g.DrawLine(new Pen(Color.Black), 50, 300, 550, 300);
    g.DrawString("(年份)", new Font("宋体", 10), new SolidBrush(Color.Black),
      new Point(50, 300));
    //绘制 Y 轴
    g.DrawLine(new Pen(Color.Black), 50, 50, 50, 300);
    g.DrawString("(访问量)", new Font("宋体", 10), new SolidBrush(Color.Black),
      new Point(10, 290));
}
```

然后在 X 轴绘制月份文本(1~6)，在 Y 轴绘制访问量文本(10000~40000)。该部分的代码如下所示：

```
//绘制月份
int startMonth = 1;
int posNum = 100;
for (int i=0; i<6; i++)
{
    string strmonth = startMonth+"月";
    g.DrawString(strmonth, new Font("宋体", 12), new SolidBrush(Color.Black),
      new Point(posNum, 300));
```

```
    startMonth = startMonth + 1;
    posNum = posNum + 80;
}
//绘制访问量
float startNum = 40000;
int posNum2 = 90;
for (int i=0; i<4; i++)
{
    string strNum = startNum.ToString();
    g.DrawString(strNum, new Font("宋体", 12), new SolidBrush(Color.Black),
      new Point(5, posNum2));
    startNum = startNum -10000;
    posNum2 = posNum2 + 50;
}
```

接下来，绘制矩形以表示访问量。在这里需要注意的是矩形区域的起点坐标以及各区域之间的间距。然后就是使用不同颜色进行填充。这部分代码如下：

```
//绘制矩形以表示访问量
int drawPosNum = 100;
Random rd = new Random();
for (int i=0; i<6; i++)
{
    //绘制红色矩形
    float flNumber1 = rd.Next(100, 150);
    g.DrawRectangle(
      new Pen(Color.Red), drawPosNum, 300 - flNumber1, 10, flNumber1);
    SolidBrush sb1 = new SolidBrush(Color.Red);
    g.FillRectangle(sb1, drawPosNum, 300 - flNumber1, 10, flNumber1);
    //绘制蓝色矩形
    float flNumber2 = rd.Next(80, 100);
    g.DrawRectangle(
      new Pen(Color.Blue), (drawPosNum + 15), 300 - flNumber2, 10, flNumber2);
    SolidBrush sb2 = new SolidBrush(Color.Blue);
    g.FillRectangle(sb2, (drawPosNum + 15), 300 - flNumber2, 10, flNumber2);
    //绘制绿色矩形
    float flNumber3 = rd.Next(180,200);
    g.DrawRectangle(
      new Pen(Color.Green), (drawPosNum + 30), 300 - flNumber3, 10, flNumber3);
    SolidBrush sb3 = new SolidBrush(Color.Green);
    g.FillRectangle(sb3, (drawPosNum + 30), 300 - flNumber3, 10, flNumber3);
    drawPosNum = drawPosNum + 80;
}
```

在上述代码中，我们将 Random 实例化后，通过它的 Next()方法来获取相应的数据以确定所要绘制矩形的高度。而在实际的应用中，该高度需要用从数据库中获得的数据与 Y 轴所能表示的高度进行某种计算得到。

下面，我们再来绘制三个小矩形，用来说明一下每个不同颜色的矩形所代表的意义。该部分代码如下：

```
//绘制红色小矩形
g.DrawRectangle(new Pen(Color.Red), 50, 330, 30,10);
SolidBrush sb4 = new SolidBrush(Color.Red);
g.FillRectangle(sb4, 50, 330, 30, 10);
//绘制蓝色小矩形
g.DrawRectangle(new Pen(Color.Blue), 200, 330, 30, 10);
SolidBrush sb5 = new SolidBrush(Color.Blue);
g.FillRectangle(sb5, 200, 330, 30, 10);
g.DrawString("日平均访问量", new Font("宋体", 10), new SolidBrush(Color.Black),
  new Point(250, 330));
//绘制绿色小矩形
g.DrawRectangle(new Pen(Color.Green), 350, 330, 30, 10);
SolidBrush sb6 = new SolidBrush(Color.Green);
g.FillRectangle(sb6, 350, 330, 30, 10);
//绘制文本说明
g.DrawString("日最高访问量", new Font("宋体", 10), new SolidBrush(Color.Black),
  new Point(100, 330));
g.DrawString("日最高访问量", new Font("宋体", 10), new SolidBrush(Color.Black),
  new Point(100, 330));
g.DrawString("当月访问总量", new Font("宋体", 10), new SolidBrush(Color.Black),
  new Point(400, 330));
```

最后，我们创建一个输出流，将图片输出，并释放资源，该部分代码与上一节相同，这里就不再重复。经过以上代码的编写，我们就完成了一个柱状图的绘制。接下来，我们新建一个 Default.aspx 网页，并在合适位置通过如下的代码将饼状图引入页面中：

```
<img alt="柱状图" src="ColumnChart.aspx"/>
```

10.9.4 运行结果

当 Default.aspx 网页设计完成后，我们运行一下页面，在浏览器中看看忙活了半天的效果如何。如图 10-21 所示，效果还真是不错！



图 10-21　绘制柱状图的效果

10.9.5　实例分析



源码解析：

本实例应用到 DrawString()、DrawRectangle()和 FillRectangle()等几个常用的绘图方法。其中 DrawString()方法我们已经再熟悉不过了。DrawRectangle()和 FillRectangle()方法的使用其实也不难，关键的问题在于它们起点坐标的定位。

从实例的代码中看到，第一个红色矩形的起点坐标为(100, 300−flNumber1)，其中 X 轴点为 100，这个好理解，那么 Y 轴点为 300−flNumber1 呢，为什么不是 360−flNumber1 呢？其实 300 为下面 X 轴线到画布上边框的距离，要想矩形的高度为 flNumber1 并且矩形的下边框刚好和下面的 X 轴线重合，那么 Y 轴点的值只能为 300−flNumber1。

10.10　常见问题解答

10.10.1　ASP.NET绘图显示的位置



ASP.NET 绘图显示的位置。
网络课堂：http://bbs.itzcn.com/thread-9863-1-1.html

在 CS 代码里绘制了一个位图，如果用 g.Save(Response.OutputStream, ImageFormat.Gif);输出图片，这个图片显示在网页的左上角，而且原来的网页内容也不见了。请问一下，如何在指定的位置显示这个位图呢？最好是能成功运行的代码。

【解决办法】：这很简单。

先在前台用个<img>标签占位，再通过后台给它指定要显示的路径，也就是你要显示的图片。运行的时候就可以完全当一个 img 标记来操作就行了，你可以使用 CSS 设置它的大小、位置等。

10.10.2　ASP.NET用Bitmap画图在Firefox中显示为乱码



ASP.NET 用 Bitmap 画图在 Firefox 中显示为乱码。
网络课堂：http://bbs.itzcn.com/thread-9865-1-1.html

ASP.NET 用 Bitmap 画图在火狐浏览器中显示为乱码，在 IE 中可以正确显示啊，应该怎么办呢？就不能用这个类了吗？

【解决办法】：如果你是使用 GDI+在后台代码中绘图，则与浏览器无关的。问题可能不是出现这个地方，或许是你程序编码的问题。

不过如果说在绘图的时候没有设置输入格式，可能会出现这个问题。你可以使用下面的代

码指定程序输出类型：

```
response.ContentType = "image/gif"
```

10.11 习　　题

一、填空题

(1) ________类表示一个与设备无关的绘图基类，它能够实现很多基本图形的绘制，它是 GDI+的基础。

(2) Pen 类用于定义绘制直线和曲线的对象，它有若干属性，其中用于描述实线、曲线、点划线等的是________属性。

(3) Graphics 类的 DrawLin()方法以________结构的变量作为参数。________结构存储 4 个整数，用于表示矩形的位置和大小。

(4) 要绘制一幅图像，需要首先将该图像文件加载到内存中，这可通过 Image 类的________方法实现。

(5) 在下面的空白处填写合适的代码，使其可以使用红色、黑体、20 号来绘制“你好吗”文字：

```
Font f = new __________;
Brush b = new SolidBrush(__________);
String addText = "你好吗";
g.DrawString(__________, f, b, 20, 20);
```

二、选择题

(1) GDI+的坐标系统中原点是基于页面的________。

A. 右上点　　B. 左下点　　C. 左上点　　D. 右下点

(2) 在 System.Drawing 命名空间中有很多常用的 GDI+结构，下列选项中________不是 System.Drawing 命名空间中的结构。

A. Color　　B. Point　　C. Rectangle　　D. Font

(3) 画刷可用来填充区域，其中________可用指定的 Image 图像或图像的一部分填充区域。

A. 线性渐变画刷(LinearGradientBrush)

B. 阴影画刷(HatchBrush)

C. 单色画刷(SolidBrush)

D. 纹理画刷(TextureBrush)

(4) 绘制文本需要使用 DrawString()方法，下列哪一项不是使用该方法时必须有的参数？________

A. 要绘制的字符串　　B. 绘制字符串所使用的字体

C. 输出文本的格式化属性　　D. 绘制字符串所使用的画刷

(5) 关于 Size 结构和 Point 结构下列描述错误的是__________。

A. 若有“Point point=Point(20,5)+Size(-15,5)”语句，执行的结果为：Point(5，10)

B. 若将 Point 结构变量和 Size 结构变量相减，会返回 Size 结构变量

C. 若将两个 Size 结构变量相减，则是将它们的 Width 和 Height 属性的值对应相减

D. 若有“Point point=Point(20,5)-Size(-10,-5) 语句，执行的结果为：Point(30，10)

(6) Graphics 类提供的__________方法可以绘制一个多边形。

A. DrawPolygon　　B. DrawClosedCurve

C. FillPolygon　　D. FillClosedCurve

(7) 下面的代码使用 DrawArc()方法绘制了一个图形，运行结果为__________。

```
Pen ptrPen = new Pen(Color.Red, 2);
pGraph.DrawArc(ptrPen, 10, 10, 60, 60, 0, 360);
```

A. 一个椭圆形

B. 一个正圆形

C. 一个规则半圆形

D. 一个实心圆形

三、上机练习

上机练习 1：绘制汽车标志。

汽车标志对读者来说也许并不陌生，它用于区分不同生产厂家的不同品牌的汽车。例如，宝马、奔驰、大众等各有自己的汽车标志。本次练习，要求利用 DrawEllipse()方法和 DrawPie()方法绘制如图 10-22 所示的汽车标志效果。

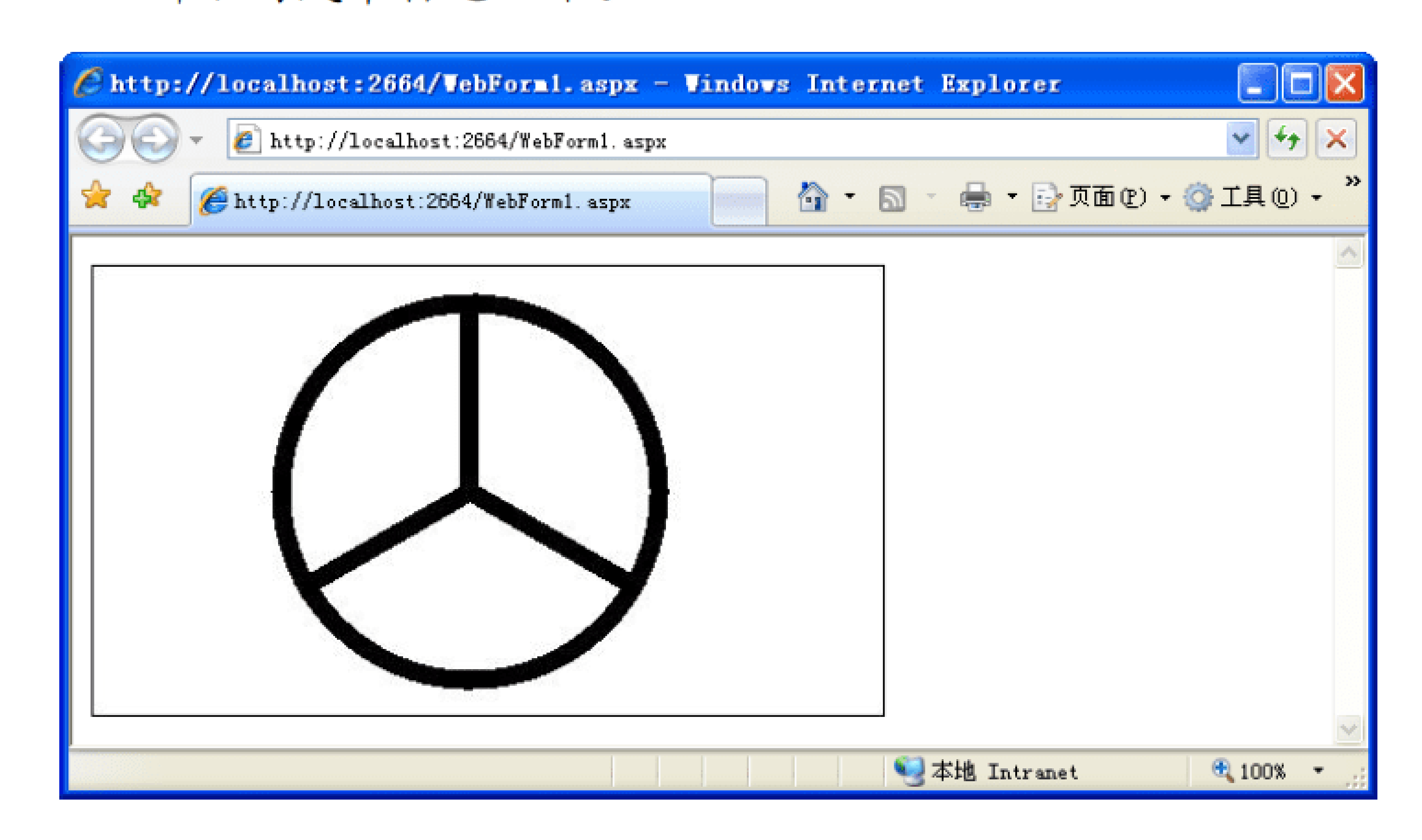


图 10-22　汽车标志效果

上机练习 2：在页面绘制古诗。

通过本章的学习，相信读者对如何使用 GDI+在页面上进行绘图有了更多的了解。本次练习的重点是 DrawString()方法，通过它将一首古诗显示到页面，并使用阴影画刷 HatchBrush 进行填充，最终运行效果如图 10-23 所示。

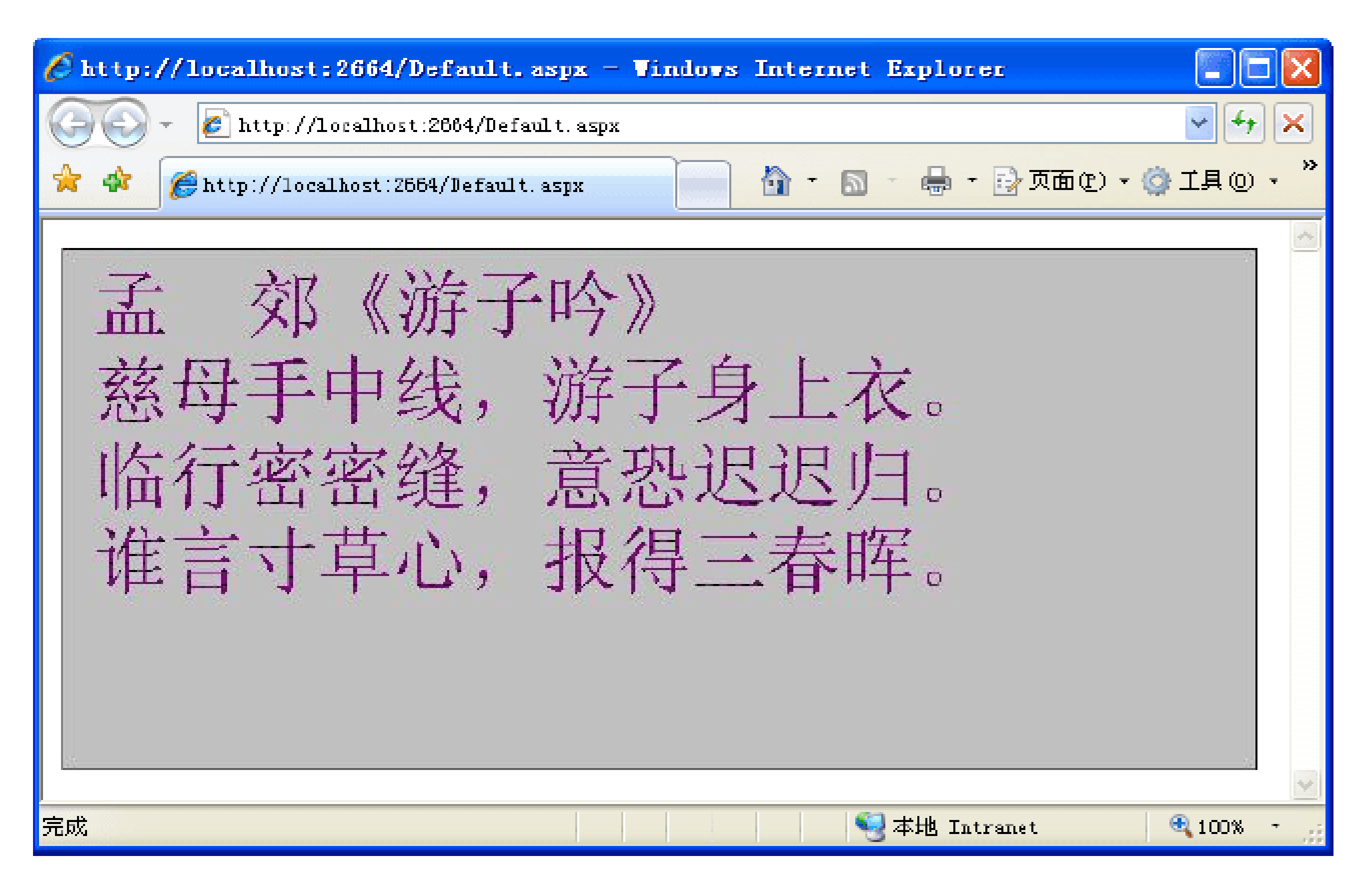


图 10-23　绘制古诗的效果



第 11 章　获取服务器端的秘密

内容摘要：

基于 HTTP 协议的特性，将 Web 分为客户端与服务器端。在客户端经常需要与服务器端进行信息通信，以及获取服务器端的状态信息，例如从客户端提交一个文件到服务器，将客户端的输入保存到服务器，查看服务器上的目录/文件信息等。

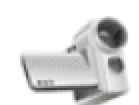
作为运行于服务器端的 ASP.NET，它能够很好地完成这些任务，而且还提供了很多操作方法，如打开和关闭文件、读取和写入文件内容、查看文件的属性、读取和获取目录的属性，还有文件的上传等。

学习目标：

- 了解 System.IO 命名空间
- 掌握 File 和 FileInfo 类的应用
- 熟练运用对文件的读写操作
- 理解文件流对文件操作时的作用与影响
- 熟悉获取文件、目录和驱动器信息的方法
- 掌握 FileUpload 控件上传文件的过程

11.1　读取文件内容

操作文件基本上是每个应用程序都必须做的事情。在 ASP.NET 中，对文件操作的类都位于 System.IO 命名空间，如 File、FileInfo、FileStream、Directory 和 DriveInfo 类等。其中使用最多的应该是 File 类，它负责从文件中读取内容，可以是一个、一行或者全部。

视频教学：光盘/videos/11/Read&Write.avi　　长度：5 分钟

11.1.1　基础知识——StreamReader类

在读取文件之前，首先要确认文件是否存在。这可以使用 ASP.NET 中 File 类的 Exists()方法来完成，该类位于 System.IO 命名空间下，如下是 Exists()方法的定义：

```
public static bool Exists(string path)
```

参数 path 表示一个要检查文件的路径。如果在 path 中包含指定的文件，则返回 true，否则为 false。如果 path 为空引用、无效路径或零长度字符串，此方法也将返回 false。



如果调用时不具有读取指定文件所需的足够权限，则不引发异常并且该方法返回 false，这与 path 是否存在无关。

StreamReader 类可以读取各种基于文本的文件，它会以一种特定的编码从字节流中读取字符，还可以读取文件的各行信息。StreamReader 类最常用的有 4 个方法。

- Read：读取输入流中的下一个字符或下一组字符，没有可用时则返回-1。
- ReadLine：从当前流中读取一行字符并将数据作为字符串返回，如果到达了文件的末尾，则为空引用。
- ReadToEnd：读取从文件的当前位置到文件结尾的字符串。如果当前位置为文件头，则读取整个文件。
- Close：关闭读取流并释放资源，在读取完成后调用。

StreamReader 默认采用 UTF-8 作为读取编码，而不是当前系统的 ANSI 编码。因为，UTF-8 可以正确处理 Unicode 字符并提供一个一致的结果。此外，也可以在 StreamReader 的构造函数中指定其他编码。如下给出了常用的两种构造函数形式：

```
//为 String 指定的文件名初始化流
StreamReader(String)
//用 Encoding 指定的编码来初始化 String 读取流
StreamReader(String, Encoding)
```

11.1.2　实例描述

岁月匆匆，时光如箭，留下的只有那回忆带给我们的无限长思。寂静的夜晚，敲打键盘记

述岁月流逝！感叹，搞 IT 技术编程的，注定很寂寞。

准备休息时，看到右下角闪动着一个陌生的头像。原来是我的大学同学小祺，三言两语的寒碜之后，知道他开设了一个书店(汗，人家现在是自己创业，啥子时候俺也……)。交谈时说到，随着图书种类和量的增加，如何有效地管理这些图书则成为重要的问题。

我的建议是采用现代化高科技进行数字化，通过一种软件来对这些图书进行追踪。经过一番的讨论，最后他说“老同学，你是这方面的行家，这个软件就交给你了，帮忙实现一下。”

哎，本来工作就很忙，没有多少时间休息，现在就更忙了，这不是找事吗！

躺在床上还考虑着，其实并不是多复杂，连数据库都不用(偷工减料)，可直接读取文件。每个文本文件代表一本书，里面是书的简介，将书名作为文件的名称进行保存。然后为了方便管理，所有文件都保存到一个特定的目录内。

有了思路，做起来就简单了。更何况对于一个 ASP.NET 专业人员来说，更是小菜一碟，三下五除二就出来模型了。

11.1.3　实例应用

【例 11-1】读取文件内容。

读取文件内容的具体实现过程和操作步骤如下。

(1)　首先在 VS2008 中新建一个 ASP.NET 网站，并在网站中新建 resource 子目录作为保存的根目录。

(2)　向测试目录 resource 中添加一个测试文件，并补充一些内容。这里使用了国学“三字经.txt”文件，完整路径是“网站/resource/三字经.txt”。

(3)　完成准备工作之后，在网站新建一个 ASPX 文件并添加一个 Button 和一个 Literal 控件，最终代码如下：

```
<p>
<asp:Button ID="btnRead" runat="server" Text="读取内容" Width="100px"
  OnClick="btnRead_Click" />
<br />
<asp:Literal ID="Literal1" runat="server"></asp:Literal>
</p>
```

(4)　使用按钮模拟读取的触发事件，Literal 控件用来显示结果。这一步编写按钮单击后要执行的操作，也就是读取文件的内容并显示。具体如下：

```
protected void btnRead_Click(object sender, EventArgs e)
{
    string fileURL = Server.MapPath("resource/三字经.txt"); //文件路径
    if (File.Exists(fileURL))                               //判断文件是否存在
    {                                                       读取文件
        StreamReader rdFile = new StreamReader(fileURL, Encoding.UTF8);
        string content = rdFile.ReadToEnd();         //获取内容
        rdFile.Close();                              //关闭文件
        Literal1.Text = content;                     //显示内容
    }
```

```
    else
    {
        Literal1.Text = "没有找到文件";                    //提示错误
    }
}
```



在本实例中需要引用 System.IO 和 System.Text 两个命名空间。它们同样适用于本章的其他实例。

11.1.4　运行结果

运行 ASPX 页面，再单击“读取内容”按钮，此时 ASP.NET 将会读取“resource/三字经.txt”文件并在下方的 Literal 控件中显示文件内容。最终运行效果如图 11-1 所示。



图 11-1　读取文件内容的效果

11.1.5　实例分析



源码解析：

利用空闲间隙，这么一个能够实现读取功能的雏形就诞生了。

从本实例中可以看出 StreamReader 类的使用步骤为：先声明一个 StreamReader 类的对象，在构造函数中声明文件路径和编码。这一步将打开文件，便可以从文件中读取数据。然后调用 StreamReader 类的方法读取数据。最后完成时调用 Close()方法关闭读取流。

如果指定的文件不存在，File.Exists()方法将返回 False，并提示“没有找到文件”。

11.2　文件信息保存

文件作为一些文本信息的存储载体，除了可以进行读取之外，还可以将指定内容保存到文件，也就是写入文件。在文件写入时，要考虑文件是否存在，是否只读，以覆盖还是追加方式

写入等，这也是本节要介绍的。

视频教学：光盘/videos/11/StreamWrite.avi　　长度：5 分钟

11.2.1　基础知识——StreamWriter类

要在一个文件中保存信息，前提是必须具有文件的写入权限。ASP.NET 中 StreamWriter 类可以以流的方式用一种特定的编码向文件中写入字符。

下面是该类最常用的方法。

- Write：将字符串写入文件。
- WriteLine：向文件写入一行字符串，也就是说在文件中写入字符串并换行。
- Flush：清理当前编写器的所有缓冲区，并将缓冲区数据写入文件。
- Close：关闭写入流并释放资源，应在写入完成后调用以防止数据丢失。

StreamWriter 类的一般使用步骤为：先实例化一个 StreamWriter 对象，然后调用它的方法将字符流写入文件中，最后调用 Close()方法保存写入的字符并释放资源。

如下给出了第 1 步实例化 StreamWriter 对象时常用的 3 种构造函数形式：

```
//用 UTF-8 编码为指定的 Stream 流做初始化
StreamWriter(Stream)
//使用默认编码为 String 指定的文件做流初始化
StreamWriter(String)
//用指定的 Encoding 编码来初始化 Stream 流
StreamWriter(Stream, Encoding)
//使用指定 Encoding 编码为 String 指定的文件做流初始化，Boolean 标识是否向文件中追加内容
StreamWriter(String, Boolean, Encoding)
```



在实例化 StreamWriter 对象时，如果指定的文件不存在，构造函数会自动创建一个新文件。存在时，可选择是改写还是追加内容。

11.2.2　实例描述

使用百度搜索时经常会在结果中看到来自百科的内容。点进去之后，可以看到该关键字的详细内容。如何不用数据库实现对这个关键字与内容的保存呢？

想到这个题目，先记录到教案上，将它作为本节课留给学生的动手练习。随后，自己动手把这个简单的小程序给实现。

具体的描述如下。

制作一个页面，为用户提供输入“标题”和“描述”的文本框。单击按钮之后，将这些内容分行写入文件中。要求：

- 第一行是标题的内容。
- 第二行是描述的内容。
- 文件保存路径是站点下的 resource 目录，名称为 new_word.txt。

11.2.3 实例应用

【例 11-2】文件信息保存。文件信息保存的具体实现过程与操作步骤如下。

(1) 创建一个 ASPX 页面，添加一个 TextBox 作为“标题”输入框，设置 ID 为 inputTitle。

(2) 再添加一个 TextBox 作为“描述”输入框，将 TextMode 设为 MultiLine，允许接收多行文本，ID 设置为 inputText。

(3) 添加一个 Button 控件，用于提交。最终布局代码如下：

```
<p>标题：<asp:TextBox ID="inputTitle" runat="server"
Width="150px"></asp:TextBox><br />
描述：<asp:TextBox ID="inputText" runat="server" Height="70px"
  TextMode="MultiLine" Width="470px"></asp:TextBox><br />
<asp:Button ID="btnSave" runat="server" Text="保存内容" Width="100px"
  OnClick="btnSave_Click" /></p>
```

(4) 双击“保存内容”按钮进入后台.cs 文件，编写实现代码，具体如下：

```
protected void btnSave_Click(object sender, EventArgs e)
{
    string fileURL = Server.MapPath("resource/new_word.txt");  //获取文件路径
    string words = inputText.Text;                              //获取输入的标题
    string title = inputTitle.Text;                             //获取输入的描述
    StreamWriter sw =
      new StreamWriter(fileURL, false, Encoding.UTF8);          //创建文件的写入流
    sw.WriteLine(title);                                        //写入一行标题
    sw.WriteLine(words);                                        //写入一行描述
    sw.Close();                                                 //关闭写入流
}
```

11.2.4 运行结果

实例运行后的效果类似图 11-2 所示，可以看到在最终的 new_word.txt 文件内，分两行显示了输入的标题和描述。

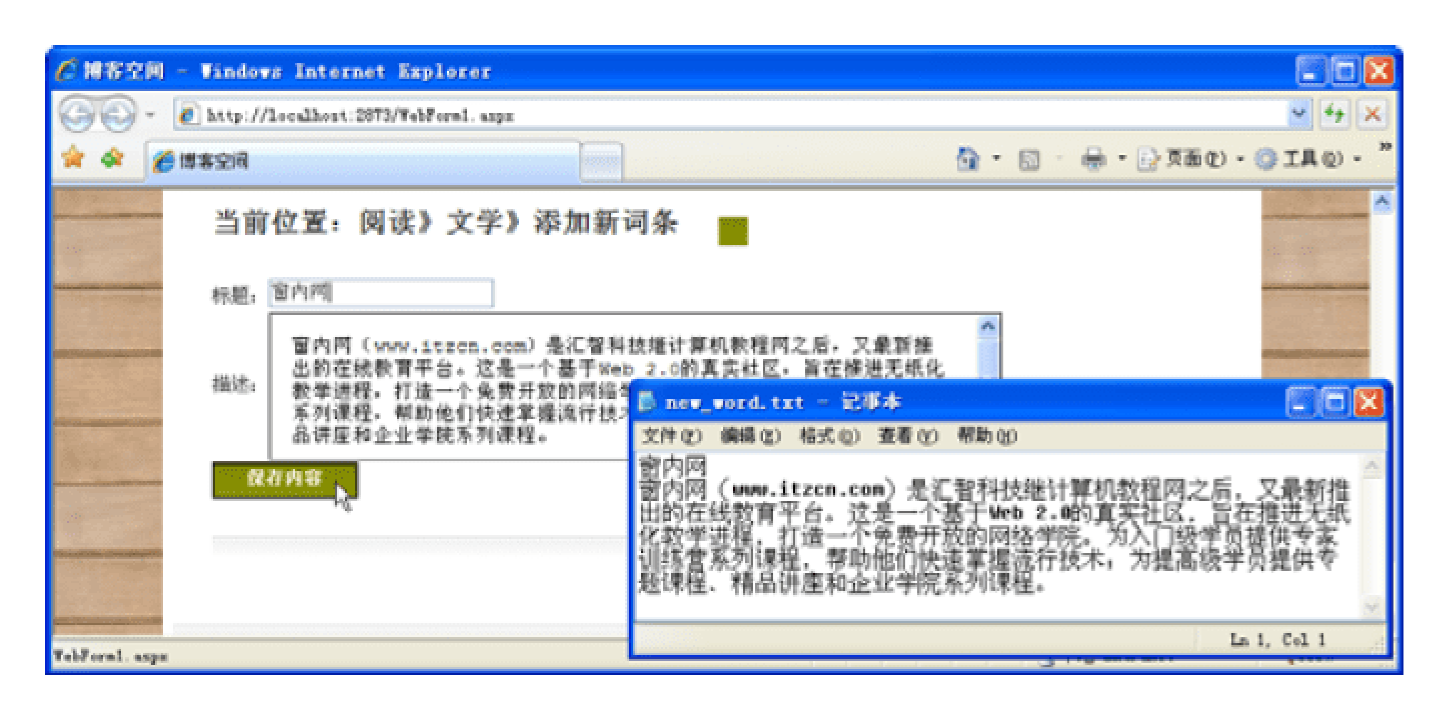


图 11-2 保存信息到文件的效果

11.2.5　实例分析



源码解析：

在代码中初始化一个 StreamWriter 类实例 sw。通过构造函数的指定，sw 会判断是否存在 resource/new_word.txt 文件，没有则创建该文件；Encoding.UTF8 指定写入文件时采用的编码。

再往下调用 WriteLine()方法向文件中写入了两个新行，完成后调用 Close()方法关闭写入流。运行本实例会发现，每次都会替换先前的内容。

如果希望向文件中追加内容，则可以将 StreamWriter 构造函数中的 false 改为 true。这样，每一次输入的内容将追加到 new_word.txt 文件最后。

11.3　浏览文件信息

System.IO 命名空间下的 FileInfo 类同样可以操作文件。与 File 类相比，它是一个实例类，需要实例化才能使用，因此仅进行一次安全检查，而且提供了更多用于操作文件的方法。例如，获取文件的扩展名、大小以及修改时间等信息。



视频教学：光盘/videos/11/FileInfo.avi　　　　长度：3 分钟

11.3.1　基础知识——FileInfo类

FileInfo 类是操作文件时使用最多的类之一，它可以用来创建、复制、删除、移动和打开文件等，而且还能创建 FileStream 对象。

作为实例对象操作的类，FileInfo 提供了很多属性来获取文件的相关信息，例如文件大小、文件创建时间和最后一次更新时间等，如表 11-1 所示。

表 11-1　FileInfo类的文件相关属性

属　性	说　明
Attributes	获取当前从 FileSystemInfo 继承的 FileAttributes 属性
CreationTime	获取当前 FileSystemInfo 对象的创建时间
Directory	获取父目录的实例
DirectoryName	获取表示目录的完整路径的字符串
Exists	获取指示文件是否存在的值
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
IsReadOnly	获取或设置确定当前文件是否为只读
LastAccessTime	获取或设置上次访问当前文件或目录的时间

续表

属　性	说　明
LastWriteTime	获取或设置上次写入当前文件或目录的时间
Length	获取当前文件的大小
Name	获取文件名

例如，要查看 C:\Boot.ini 文件的大小，代码如下：

```
FileInfo fi = new FileInfo(@"C:\boot.ini");
Response.Write("Boot.ini 文件的大小为:" + fi.Length);
```

如果打算多次重用某个对象，可考虑使用 FileInfo 的实例方法，而不是 File 类的相应静态方法，因为并不总是需要安全检查。

默认情况下，FileInfo 类将向所有用户授予对新文件的完全读/写访问权限。另外，如果要多次重用文件对象，可以调用 FileInfo 类的实例方法，而不是 File 类的相应静态方法，因为安全检查并不每次都需要。在表 11-2 中列出了该类的一些实例方法。

表 11-2　FileInfo类实例方法

方　法	说　明
AppendText	创建一个 StreamWriter，向 FileInfo 的此实例表示向文件追加文本
CopyTo	将现有文件复制到新文件
Create	创建文件
CreateText	创建写入新文本文件的 StreamWriter 对象
Delete	删除指定文件
Encrypt	将某个文件加密，使得只有加密该文件的账户才能将其解密
MoveTo	将指定文件移到新位置，并提供指定新文件名的选项
Open	打开指定文件
OpenRead	创建只读 FileStream 对象
OpenText	创建使用 UTF8 编码、从现有文本文件中进行读取的 StreamReader 对象
OpenWrite	创建写入的 FileStream 对象
Replace	使用当前 FileInfo 对象所描述的文件替换指定文件的内容，这一过程将删除原始文件，并创建被替换文件的备份

例如，在 C 盘创建一个 string.txt 文件，并向文件内写入 3 行文本：

```
string path = @"c:\string.txt";              //指定文件路径
FileInfo fi = new FileInfo(path);            //实例化 FileInfo 对象
if (!fi.Exists)                          //判断文件是否存在
{
    using (StreamWriter sw = fi.CreateText())    //创建该文件
    {
        sw.WriteLine("Hello");              //写入第一行
        sw.WriteLine("And");                //写入第二行
        sw.WriteLine("Welcome");            //写入第三行
```

```
    }
}
```

11.3.2　实例描述

书店的网站已上线，可是好景不长。这不，店老板小祺那边又来电话了。说是用了一段时间后，无法得知文件的状态，不知道它是什么时候创建的，文件有多大等。

所以，希望添加一个功能，能用程序的方式读取该文件的信息，并在页面上显示。这些信息包括：文件名称、文件大小、最后更新时间、最后查看时间、是否为只读、文件完整目录、文件扩展名以及完整路径和创建时间。

11.3.3　实例应用

【例 11-3】浏览文件信息。

浏览文件信息的具体实现过程与操作步骤如下。

假设我的网站下存在一个 resource 子目录，里面包含一个“三字经.txt”文件。

(1)　新建一个 ASPX 页面，添加一个 Button 控件和一个 Literal 控件，并设置控件属性：

```
<asp:Button ID="btnRead" runat="server" Text="读取文件" Width="100px"
  OnClick="btnRead Click" />
<br />
<asp:Literal ID="Literal1" runat="server"></asp:Literal>
```

(2)　双击 Button 控件，对其按钮的单击事件进行编码，实现描述中所需的要求：

```
protected void btnRead_Click(object sender, EventArgs e)
{
    string fileURL = Server.MapPath("resource/三字经.txt"); //文件路径
    if (File.Exists(fileURL))                              //判断是否存在
    {
        FileInfo info = new FileInfo(fileURL);             //获取文件信息
        Literal1.Text = "获取到的文件名: " + info.Name + "<br/>"
          + "获取文件的大小: " + info.Length + "字节<br/>"
          + "最后一次更新时间: " + info.LastWriteTime.ToLongDateString()
          + info.LastWriteTime.ToLongTimeString() + "<br/>"
          + "最后一次查看该文件时间: " + info.LastAccessTime.ToLongDateString()
          + info.LastAccessTime.ToLongTimeString() + "<br/>"
          + "是否为只读文件: " + info.IsReadOnly.ToString() + "<br/>"
          + "文件的完整目录: " + info.FullName + "<br/>"
          + "文件扩展名为: " + info.Extension + "<br/>"
          + "目录的完整路径: " + info.DirectoryName + "<br/>"
          + "文件创建时间: " + info.CreationTime.ToLongDateString()
          + info.CreationTime.ToLongTimeString();
    }
    else
    {
```

```
        Literal1.Text = "没有找到文件。";                //提示出错
    }
}
```

使用上面的几行代码即可达成目的。

11.3.4 运行结果

运行 ASPX 页面后单击“读取文件”按钮，在下方的 Literal 控件中会显示文件的相关信息，效果如图 11-3 所示。

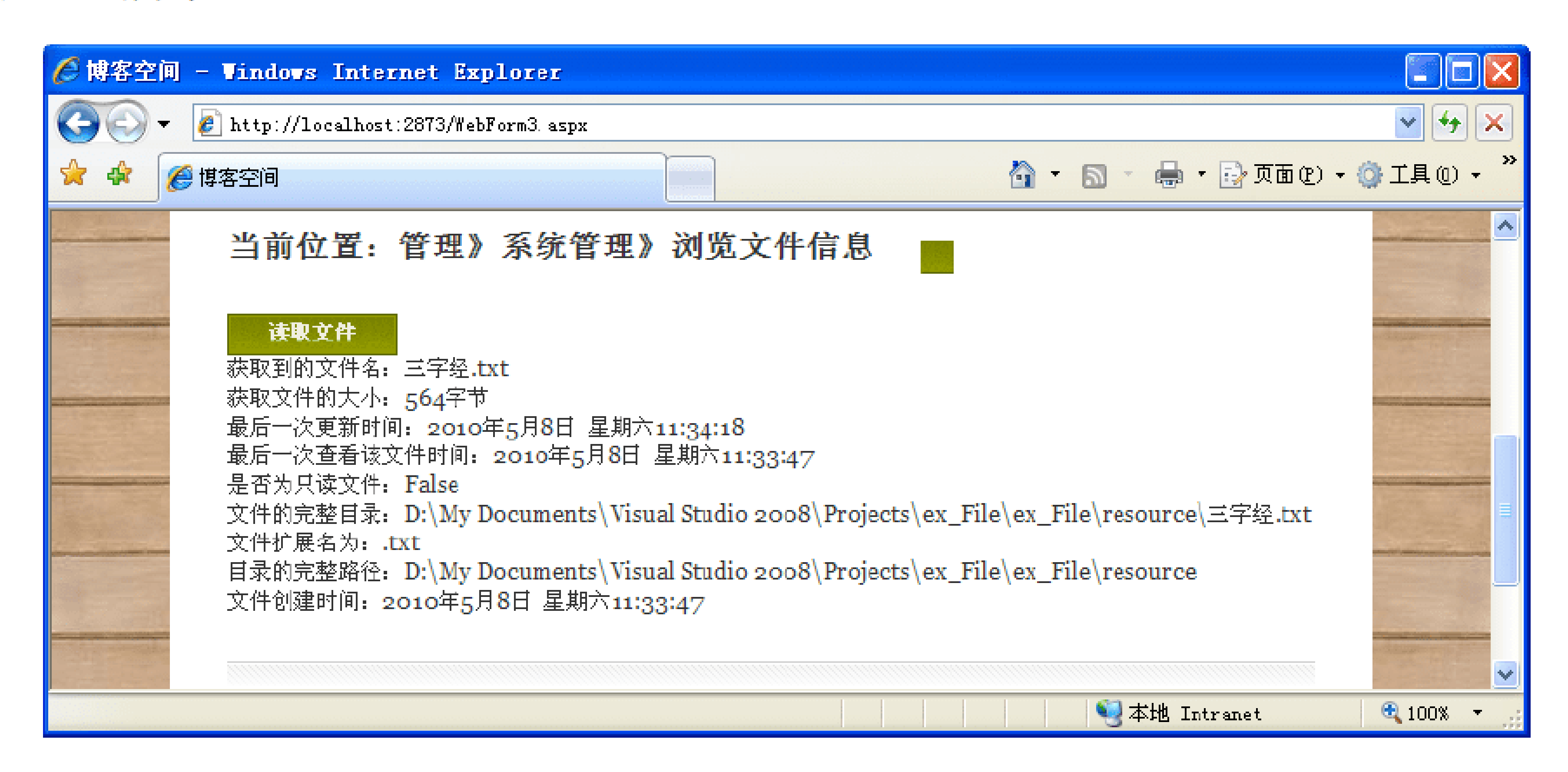


图 11-3 浏览文件信息的效果

11.3.5 实例分析



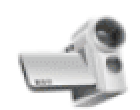
源码解析：

一个优秀的应用程序必须具有好的容错性。例如，在本实例中，虽然确保该文件已经存在，但还是对例外情况进行验证，防止运行出错。

实现时先通过 FileInfo 类的构造函数指定文件的路径，然后充分利用该类提供的属性，从文件中获取相关的信息。要注意，获取文件的时间属性时，还需要进行格式化处理。

11.4 浏览目录信息

浏览目录信息与浏览文件信息相似，都是通过 System.IO 命名空间下的类来实现的。使用目录可以对文件进行归类存储，例如，对一个网站新建一个目录，然后通过程序的方式获取目录的大小，就可以掌握网站空间是否用完。



视频教学：光盘/videos/11/OperateDirectory.avi　　　　长度：11 分钟

11.4.1　基础知识——目录操作类

在 ASP.NET 中使用 DirectoryInfo 类和 Directory 类可以对目录进行管理。它们可以完成对目录及其子目录的创建、移动、浏览等操作，甚至还可以定义隐藏目录和只读目录。

1. 以Directory类获取目录信息

Directory 是一个静态类，因此在调用其静态方法之前不必创建该类的实例。这些静态方法用于创建、移动、枚举目录和子目录等。在表 11-3 中列出 Directory 类常用的静态方法。

表 11-3　Directory类常用的静态方法

方　法	说　明
CreateDirectory	创建一个指定路径的目录
Delete	删除一个指定路径的目录
Exists	判断指定路径的目录是否存在，如果存在返回 true，否则返回 false
GetCreationTime	返回指定目录的创建时间和日期
GetLastAccessTime	返回指定目录最后一次访问的时间和日期
GetLogicalDrives	返回一个包含系统上逻辑驱动器的字符串数组
Move	将指定目录及其内容移动到新的位置
SetCurrentDirectory	指定应用程序的当前工作目录
GetCurrentDirectory	获取应用程序的当前工作目录
GetDirectories	获取指定目录下的子目录，返回为字符串数组
GetFiles	获取指定目录下的文件，返回为字符串数组

例如，判断“E:\Movie”目录是否存在，并给出相应的提示：

```
string path = @"E:\Movie";
if (Directory.Exists(path))
{
    Response.Write("<script>alert('Movie 目录存在。')</script>");
}
else
{
    Response.Write("<script>alert('找不到 Movie 目录。')</script>");
}
```

下面的代码将“E:\Music”目录下的内容移动到“E:\Movie”目录下：

```
string sourcePath = @"E:\Music";   //指定源目录
string destPath = @"E:\Movie";     //指定目标目录
Directory.Move(sourcePath, destPath); //执行移动操作
```

Move()方法的工作方式为，首先创建目标目录，然后创建目录中的内容，再将被移动的源目录删除。



在使用 Directory 类时，必须理解当前工作目录的概念。在应用程序运行时，某一时刻只能有一个当前工作目录。如果对目录或者文件使用相对目录引用，ASP.NET 将会使用当前目录作为相对引用的开始，如果使用绝对引用，则忽略当前的工作目录。

2. DirectoryInfo类获取目录信息

DirectoryInfo 类和 Directory 类一样，用于对目录进行管理。但 DirectoryInfo 类需要实例化才可以调用其方法，从而有效地对一个目录进行多种操作。DirectoryInfo 类在实例化后，可以获取目录的创建时间和最后修改时间等状态。

DirectoryInfo 类提供了 4 个属性用来获取目录的名称、父目录和根等，如下所示。

- Exists：判断指定路径的目录是否存在，如果存在返回 true，否则返回 false。
- Name：获取目录的名称。
- Parent：获取指定子目录的父目录名称。
- Root：获取目录的根部分。

此外，DirectoryInfo 类还从 FileSystemInfo 类继承了 9 个属性，如表 11-4 所示。

表 11-4　DirectoryInfo类继承的属性

属　性	说　明
Attributes	获取或设置当前目录的 FileAttributes
CreationTime	获取或设置当前目录的创建时间
CreationTimeUtc	获取或设置当前目录的创建时间，其格式为 UTC 时间
Extension	获取表示文件扩展名部分的字符串
FullName	获取目录或文件的完整目录
LastAccessTime	获取或设置上次访问当前文件或目录的时间
LastAccessTimeUtc	获取或设置上次访问当前文件或目录的时间，其格式为 UTC 时间
LastWriteTime	获取或设置上次写入当前文件或目录的时间
LastWriteTimeUtc	获取或设置上次写入当前文件或目录的时间，其格式为 UTC 时间

例如，要获取“E:\Music”目录的创建时间和最后一次访问时间：

```
string path = @"E:\Music";                          //指定路径
DirectoryInfo di = new DirectoryInfo(path);  //实例化一个 DirectoryInfo 对象
Response.Write("目录创建时间为：" + di.CreationTime.ToString());
Response.Write("最后一次访问时间为：" + di.LastAccessTime.ToString());
```

11.4.2　实例描述

利用午休的时间，为书店网站添加另一个功能，就是如何查看文件所在目录 resource 的状态信息，如目录创建时间、完整路径、最后一次访问时间、最后一次修改时间、目录名称、父目录以及所在的驱动器等。

11.4.3　实例应用

【例 11-4】浏览目录信息。

浏览目录信息的具体实现过程与操作步骤如下。

(1) 新建一个 ASPX 页面，添加一个 Button 控件和一个 Literal 控件，并设置控件属性：

```
<asp:Button ID="btnRead" runat="server" Text="读取目录" Width="100px"
  OnClick="btnRead_Click" />

<br />
<asp:Literal ID="Literal1" runat="server">
</asp:Literal>
```

这里的按钮用于单击以便查看结果，Literal 则是显示结果用的。

(2) 进入.aspx.cs 文件，为“读取目录”按钮的单击事件添加处理代码：

```
protected void btnRead_Click(object sender, EventArgs e)
{
    string directoryURL = Server.MapPath("/resource");      //指定路径

    if (Directory.Exists(directoryURL))                    //判断是否存在
    {
        DirectoryInfo info =
          new DirectoryInfo(directoryURL);   //实例化一个 DirectoryInfo 对象
        info.Attributes = FileAttributes.ReadOnly | FileAttributes.Hidden;
        Literal1.Text =
                "目录创建时间："
                + info.CreationTime.ToString()
                + "<br/>完整路径：" + info.FullName
                + "<br/>最后一次访问该目录时间：" + info.LastAccessTime.ToString()
                + "<br/>最后一次修改目录时间：" + info.LastWriteTime.ToString()
                + "<br/>目录名称：" + info.Name
                + "<br/>父目录：" + info.Parent
                + "<br/>所在驱动器：" + info.Root.ToString();
    }
    else
    {
        Literal1.Text = "目录不存在，请检验路径是否正确。";
    }
}
```

11.4.4　运行结果

运行 ASPX 页面后，单击“读取目录”按钮，在下方的 Literal 控件会显示目录的相关信息，效果如图 11-4 所示。

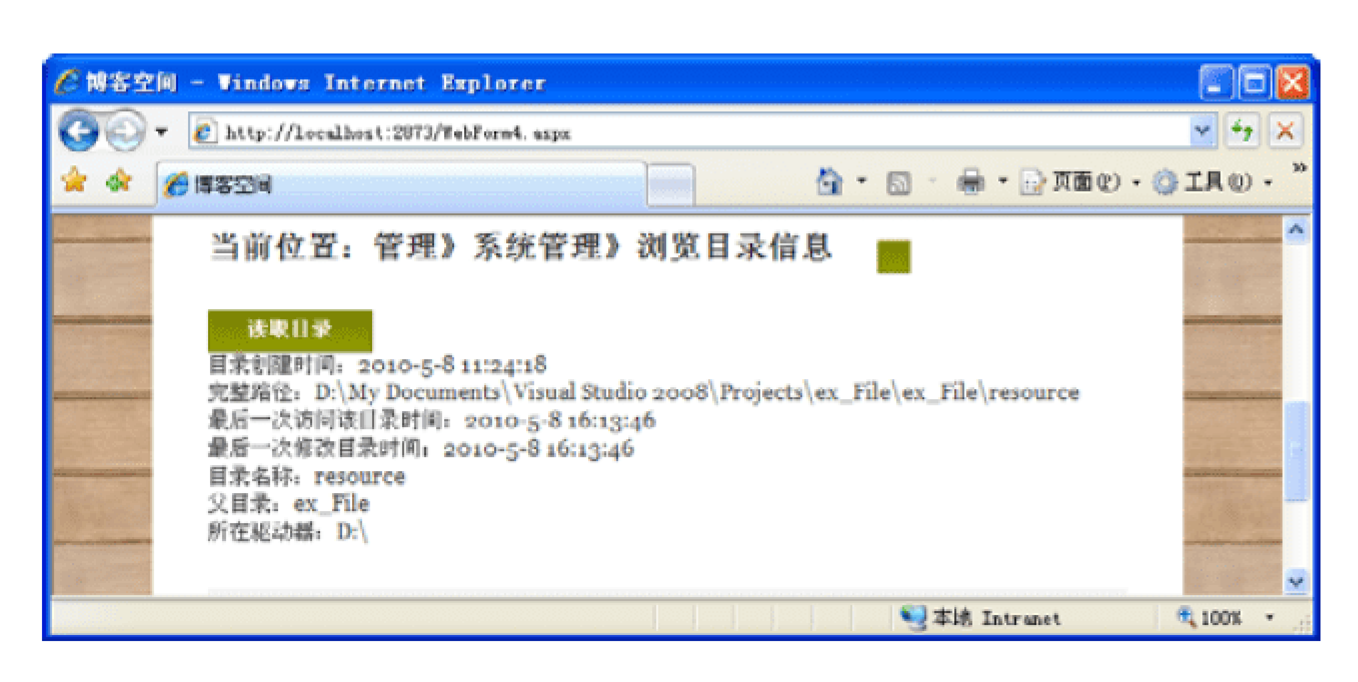


图 11-4　浏览目录信息的效果

11.4.5　实例分析


源码解析：

由于 Directory 类的静态方法都会执行安全检查，所以在这里调用它的 Exists()方法来检查是否存在 resource 目录。接下来的代码，则通过实例化一个 DirectoryInfo 对象，再调用提供的实例方法来获取要求的目录属性信息。

运行之后，将发现 resource 目录上增加了只读和隐藏属性，这说明 Attributes 属性起了作用。

11.5　浏览硬盘信息

浏览硬盘信息是指如何在客户端得知服务器上都有哪些驱动器，类型是什么(光驱或者网络驱动器)，可用空间和卷标等。通过这些信息，可以准确地分析出服务器的运行状态。


视频教学：光盘/videos/11/OperateDrive.avi　　　　长度：12 分钟

11.5.1　基础知识——DriveInfo类

System.IO 命名空间下的 DriveInfo 类可以对计算机上的驱动器进行操作。该类仅包含一个静态方法 GetDrives()，它能够检索计算机上所有逻辑驱动器的驱动器名称，并返回一个包含驱动器列表的数组。

DriveInfo 类还提供了很多与驱动器相关的属性，它们需要在实例化后使用，如下所示。

- AvailableFreeSpace：指示驱动器上的可用空闲空间量。
- DriveFormat：获取文件系统的名称，例如 NTFS 或 FAT32。
- DriveType：获取驱动器类型，如表 11-5 给出了可选值及其含义描述。
- IsReady：获取一个指示驱动器是否已准备好的值。
- Name：获取驱动器的名称。
- RootDirectory：获取驱动器的根目录。

- TotalFreeSpace：获取驱动器上的可用空闲空间总量。
- TotalSize：获取驱动器上存储空间的总大小。
- VolumeLabel：获取或设置驱动器的卷标。

表 11-5　DriveType属性

值	含义描述
Unknown	无法确定驱动器类型
Removable	可移动媒体驱动器，包括软盘驱动器和其他多种存储设备
Fixed	固定(不可移动)媒体驱动器，包括所有硬盘驱动器(包括可移动的硬盘驱动器)
Network	网络驱动器，包括网络上任何位置的共享驱动器
CDROM	CDROM 驱动器，不区分只读和可读写的 CDROM 驱动器
RAM	RAM 磁盘

例如，可用下面的代码输出 C 盘的卷标：

```
string drvName = @"C:\";                                          //指定驱动器
DriveInfo drvInfo = new DriveInfo(drvName);              //实例化一个 DriveInfo 对象
Response.Write("驱动器 C 的卷标为：" + drvInfo.VolumeLabel);        //输出卷标
```

11.5.2　实例描述

技术无止境，客户的要求也永无尽头。就说上次那个书店网站吧，现在事情又来了。除了查看网站中的文件信息、目录信息外，他们又说想看看服务器上的硬盘信息。

要求提供一个所有的驱动器列表，要用户选择一个后，输出指定驱动器的信息，包括驱动器名称、可用空间、空间总大小、驱动器类型、是否可用、根目录、可用空间以及驱动器的卷标。这个要求真是有点高了。出于安全的考虑，一般的服务器提供商都不允许网站具有访问系统的权限，而如果是本机则没有这个限制。

下面就给读者演示一下这个功能应该怎样实现。

11.5.3　实例应用

【例 11-5】浏览硬盘信息。

浏览硬盘信息的具体实现过程和操作步骤如下。

(1) 在 ASPX 页面中添加一个 ListBox 控件用于显示驱动器列表。

(2) 添加一个 Button 控件，让用户在选择驱动器后进行单击确认。

(3) 在 Button 控件下方添加一个 Literal 控件，用于显示驱动器的信息。如下所示是完成后的布局代码：

```
<table>
    <tr>
        <td valign="top">
        驱动器列表：<br />
```

```
        <asp:ListBox ID="ListBox1" class="left" runat="server"
          Width="80px" Height="90px">
        </asp:ListBox>
        </td>
        <td>
        <asp:Button ID="btnRead" runat="server" Text="读取信息"
          Width="100px" OnClick="btnRead_Click" /><br />
        <asp:Literal ID="Literal1" runat="server"></asp:Literal>
        </td>
    </tr>
</table>
```

(4) 在后台代码.cs 文件中找到 Page_Load()事件，编写在 ListBox 中显示可用驱动器列表的代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack) {                                  //是否刷新页面
        DriveInfo[] driveInfos = DriveInfo.GetDrives();  //获取所有驱动器
        foreach (DriveInfo info in driveInfos)
        {
            ListBox1.Items.Add(info.Name);                //添加到列表框
        }
    }
}
```

(5) 在按钮的单击事件中编写代码，输出要求的各种驱动器信息：

```
protected void btnRead Click(object sender, EventArgs e)
{
    //获取用户在下拉列表中选择的驱动器
    DriveInfo driveInfo = new DriveInfo(ListBox1.SelectedItem.ToString());
    //判断驱动器是否可用
    if (driveInfo.IsReady)
    {
        Literal1.Text = "驱动器名称：" + driveInfo.Name + "<br/>"
              + "驱动器可用空间：" + driveInfo.AvailableFreeSpace + "<br/>"
              + "驱动器空间总大小：" + driveInfo.TotalSize + "<br/>"
              + "文件系统的名称：" + driveInfo.DriveFormat + "<br/>"
              + "驱动器类型：" + driveInfo.DriveType + "<br/>"
              + "指示驱动器是否已准备好：" + driveInfo.IsReady + "<br/>"
              + "获取驱动器根目录：" + driveInfo.RootDirectory + "<br/>"
              + "驱动器上可用空闲空间总量：" + driveInfo.TotalFreeSpace + "<br/>"
              + "驱动器卷标：" + driveInfo.VolumeLabel;
    }
    else
    {
        Literal1.Text = "驱动器还没有准备好.";
    }
}
```

11.5.4　运行结果

运行 ASPX 页面后，首先会看到当前计算机上所有可用的驱动器列表。从中选择一项，再单击“读取信息”按钮查看结果，如图 11-5 所示。

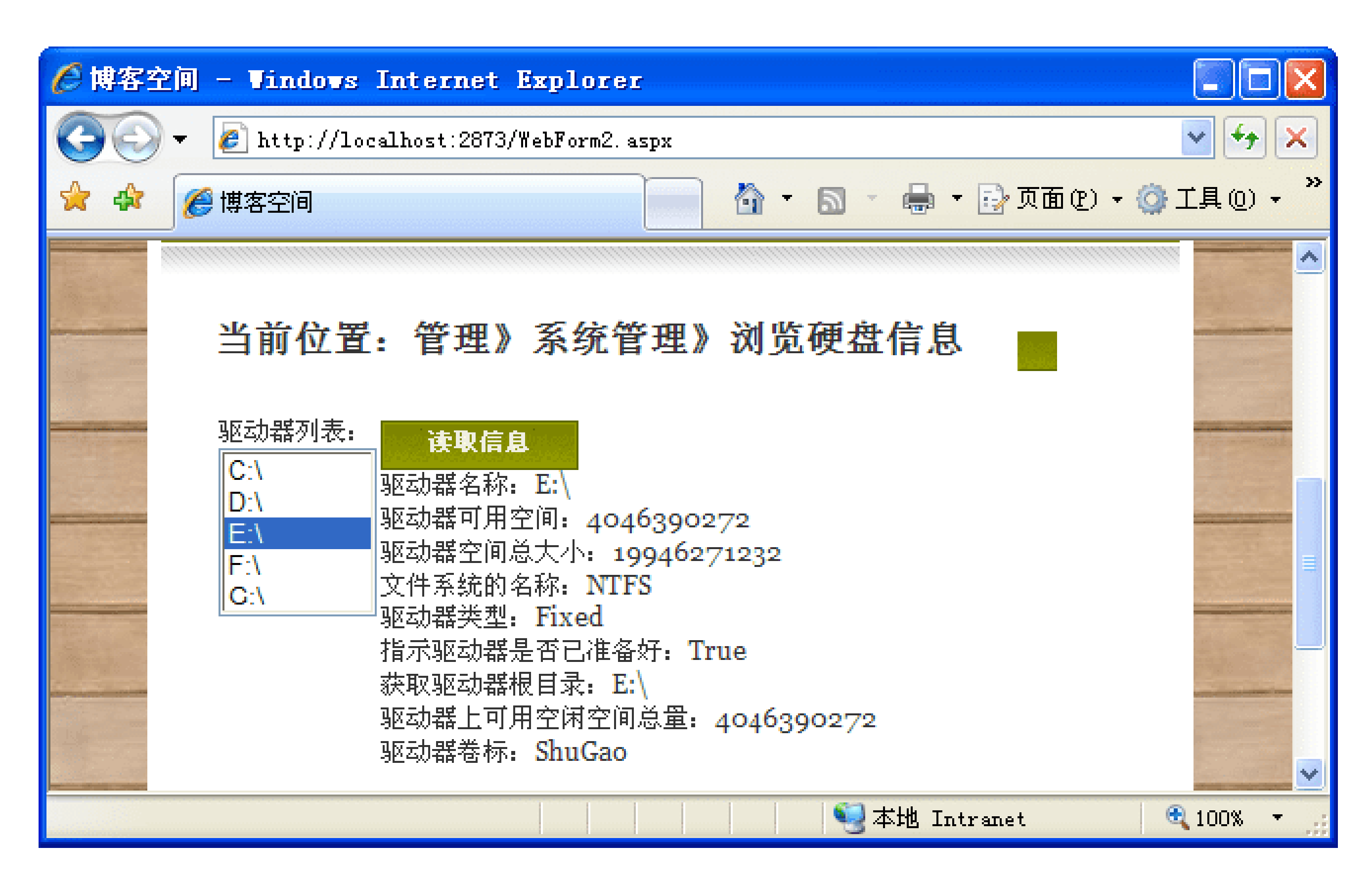


图 11-5　浏览硬盘信息的效果

11.5.5　实例分析



源码解析：

实现本实例时主要分为两步：获取驱动器列表和显示指定驱动器的详细信息。

(1) 通过调用静态方法 GetDrives()来返回一个驱动器数组，然后遍历这个数组，将每一个驱动器添加到 ListBox 中。

(2) 首先用 ListBox1.SelectedItem 来获取用户选择的驱动器，然后将此值传递到实例化 DriveInfo 类的构造函数中，接下来则调用实例属性来输出要求的各种信息。

要注意，在获取大小时默认是以字节为单位显示的。如果要以其他格式显示，还需要进行一次转换。

11.6　文 件 上 传

文件上传允许用户从客户端选择一个文件提交到服务器上进行保存。

在实际应用中使用得非常广，如上传个人简历、上传头像、上传产品照片以及上传程序代码文件等。



视频教学： 光盘/videos/11/FileUpload.avi　　长度：5 分钟

11.6.1 基础知识——FileUpload控件

文件上传对 ASP 开发来说是相当烦琐的工作，因为客户端上传文件常用加密的 Form 类型提交。而在服务器端接收这些文件也非常烦琐，数据在使用之前必须以安全的 Byte 数组的方式接收并解密。因此，大部分开发人员选择第三方组件来成该功能。

而在 ASP.NET 中上传文件变得非常简单容易，因为.NET Framework SDK 提供了一组相关的类来完成文件的上传。方法是使用 FileUpload 控件，它可以让用户浏览并选择用于上传的文件，或者输入文件的绝对路径。然后，调用该控件的 SaveAs()方法保存到服务器端。

FileUpload 控件主要有 3 个与文件上传相关的属性。

- HasFile：判断 FileUpload 控件是否包含文件，返回一个布尔值。
- PostedFile：获取要上传文件的 HttpPostedFile 对象。
- FileName：获取要上传文件的名称。

SaveAs()方法的定义如下：

```
public void SaveAs(string filename)
```

filename 参数为一个字符串，指定要在服务器上保存上传文件的完整路径。例如，将文件保存到站点的 Uploads 文件夹下，方法为：

```
// 客户端选择文件的路径
string name = FileUpload1.PostedFile.FileName;
FileInfo file = new FileInfo(name);
//获取文件名称
string fileName = file.Name;
//设置服务器端上传文件的路径
string webFilePath = Server.MapPath("Uploads/" + fileName);
// 使用 SaveAs 方法上传文件
FileUpload1.SaveAs(webFilePath);
```

11.6.2 实例描述

你方唱罢我登场。文本文件的功能都实现了，现在轮到图片文件了。在书店中用到最多的是图书封面图片，为了方便管理，要求将图片与文本文件都保存到站点的 resource 目录中，而且要对上传的文件格式进行验证，允许 BMP 格式、GIF 格式和 JPG 格式。

客户就是上帝，确定要求后开始了新一轮编码征途……

11.6.3 实例应用

【例 11-6】文件上传。

文件上传的具体实现过程及操作步骤如下。

(1) 在 ASPX 页面添加 FileUpload 控件，设置宽为 300：

```
选择要上传的图片：<br />
<asp:FileUpload ID="FileUpload1" runat="server" Width="300px" />
```

(2) 添加一个 Button 控件作为选择文件后的提交按钮，设置文本为“开始上传”：

```
<asp:Button ID="btUpload" runat="server" OnClick="btUpload_Click"
  Text="开始上传" /> <br />
```

(3) 添加两个 Label 控件，用于显示上传时的提示信息和上传后的路径，并修改 ID 为 lMsg 和 lPathInfo：

```
<asp:Label ID="lMsg" runat="server" ForeColor="red"></asp:Label><br />
上传路径：<asp:Label ID="lPathInfo" runat="server"></asp:Label><br />
```

(4) 结束对第一列的编辑，进入第二列，添加一个 Image 控件用于显示上传后的图片：

```
图片效果：<br />
<asp:Image ID="iPic" runat="server" ImageUrl="" Height="100px" />
```

(5) 在完成以上步骤后，整个页面的布局也就完成了，如图 11-6 所示为此时页面的效果。

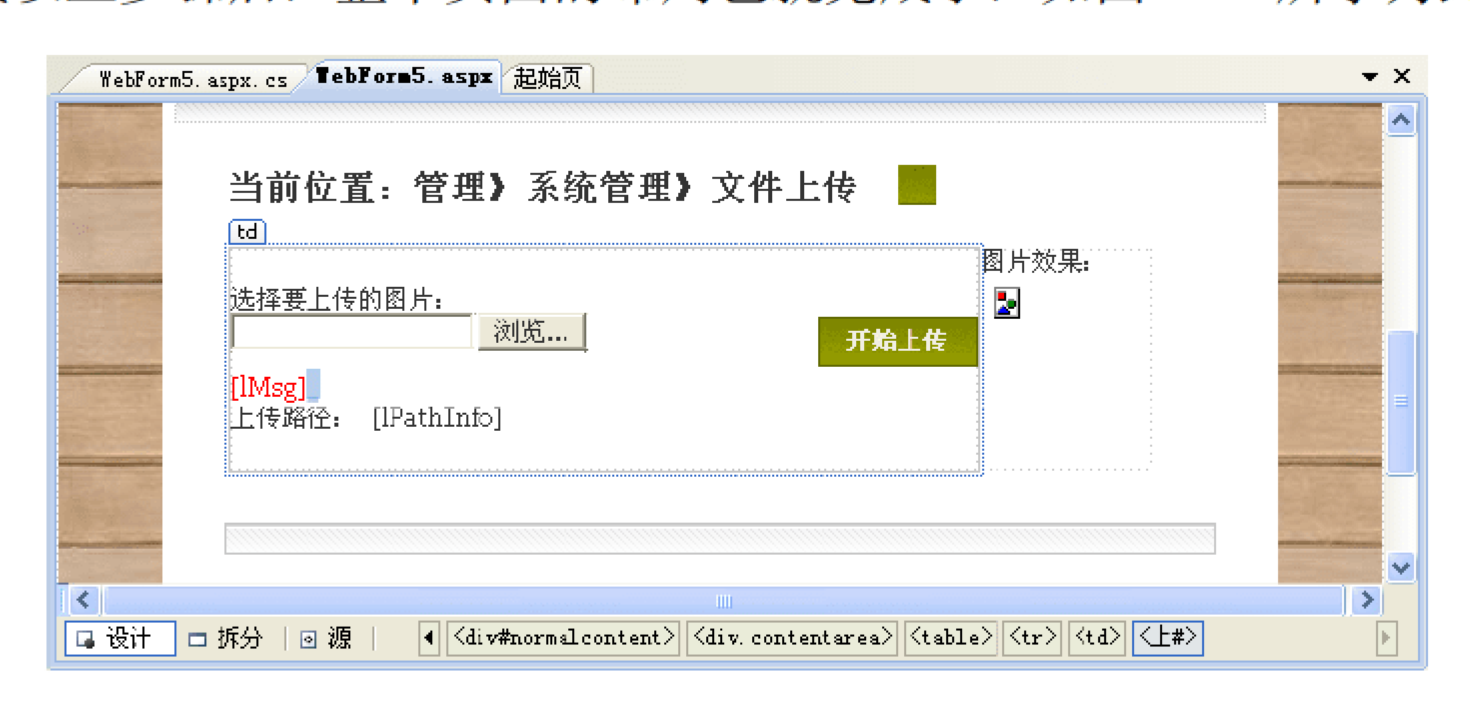


图 11-6　文件上传布局的效果

(6) 在“设计”视图中双击“开始上传”按钮，在进入的.cs 后台界面编写上传的代码，如下所示：

```
protected void btUpload_Click(object sender, EventArgs e)
{
    if (FileUpload1.HasFile)          //判断是否选择文件
    {                          //获取文件内容类型
        string fileContentType = FileUpload1.PostedFile.ContentType;
        //判断类型是否符合条件
        if (fileContentType=="image/bmp" || fileContentType=="image/gif"
          || fileContentType=="image/pjpeg")
        {
            // 客户端文件路径
            string name = FileUpload1.PostedFile.FileName;
            FileInfo file = new FileInfo(name);
            // 文件名称
            string fileName = file.Name;
            // 服务器端文件路径
            string webFilePath = Server.MapPath("resource/" + fileName);
```

```
            //判断相同文件是否存在
            if (!File.Exists(webFilePath))
            {
                try
                {
                    // 使用 SaveAs 方法保存文件
                    FileUpload1.SaveAs(webFilePath);
                    this.lMsg.Text = "提示：文件" + fileName + "上传成功！";
                    this.lPathInfo.Text = "resource/" + fileName;
                    this.iPic.ImageUrl = "resource/" + fileName;
                }
                catch (Exception ex)
                {
                    this.lMsg.Text = "提示：文件上传失败，失败原因：" + ex.Message;
                }
            }
            else
            {
                this.lMsg.Text = "提示：文件已经存在，请重命名后上传";
            }
        }
        else
        {
            this.lMsg.Text = "提示：文件类型不符";
        }
    }
}
```

11.6.4 运行结果

运行 ASPX 页面，将看到如图 11-7 所示的上传初始页面。

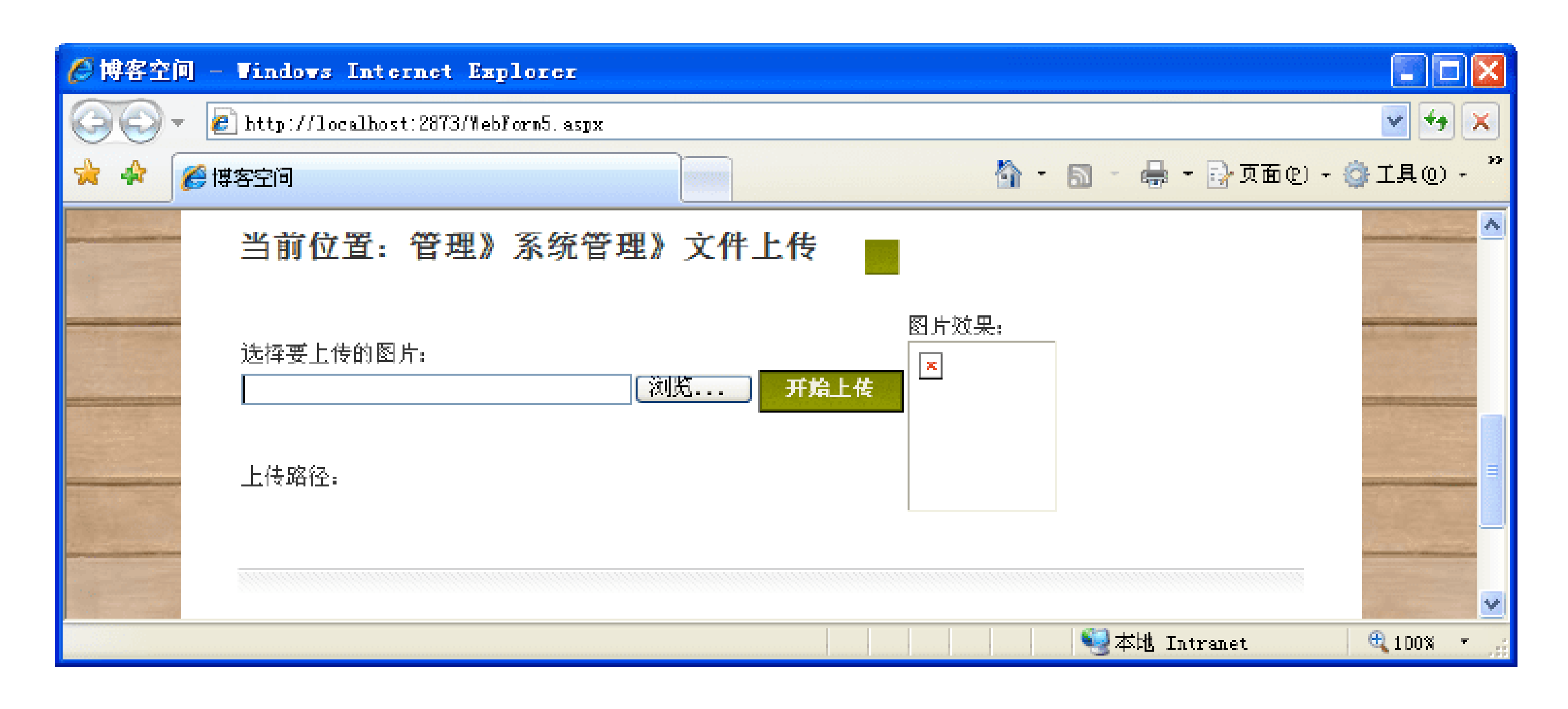


图 11-7　上传初始页面

单击页面中的“浏览”按钮，选择要上传的图片，当图片选择完成后，单击“开始上传”按钮提交到服务器。上传成功之后，将可以看到相应的提示，并可以看到上传的图片。

这里上传了公司的 Logo 图片作为测试，效果如图 11-8 所示。

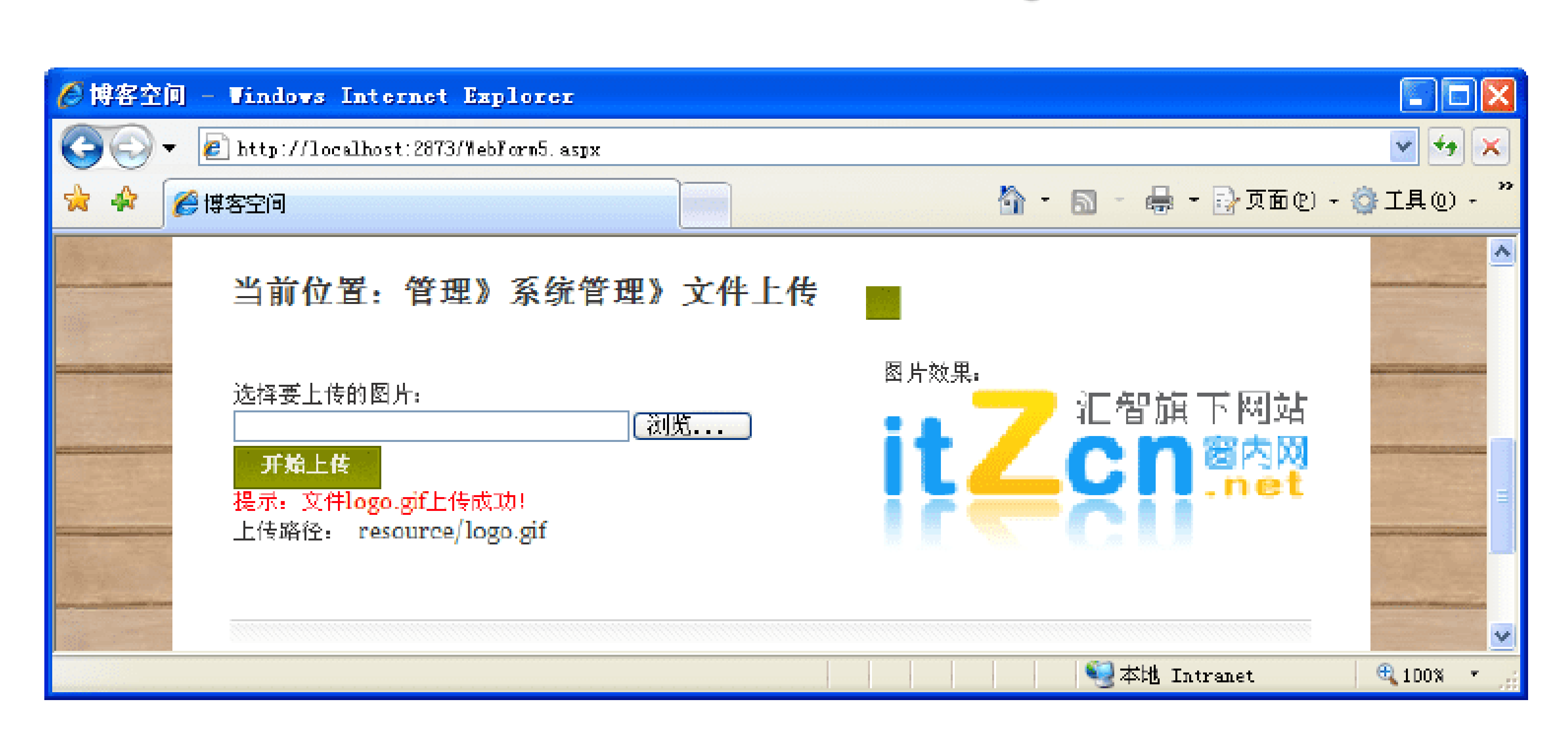


图 11-8　上传成功后的效果

11.6.5　实例分析



源码解析：

在本实例中，主要是使用了 FileUpload 控件来实现上传功能。首先，调用它的 HasFile 属性判断是否选择了文件。然后，通过 PostedFile 属性获取要上传文件的格式、文件名称等。最后调用 SaveAs()方法保存到服务器端指定的目录下，并输出相应的提示信息。

在本实例中处理了各种上传时的例外情况，如文件格式不符合条件时，存在相同文件名时以及上传失败时。这些例外情况的处理增强了程序的健壮性，并为用户提供了友好的错误提示。

11.7　文 件 下 载

在我们编写系统过程中，有很多时候需要客户端下载文件，这就需要实现文件下载功能。下载与上传是一个相反的过程，上传是将文件从客户端保存到服务器端，下载则是将文件从服务器端保存到客户端，而且下载时通常都提供一个列表，单击一个链接来完成下载过程。



视频教学：光盘/videos/11/FileDownload.avi　　　　长度：4 分钟

11.7.1　基础知识——输出文件流

文件下载时首先需要指定文件的名称和路径。再判断该文件是否存在，如果存在，则获取该文件的信息并设置到下载的流中，包括默认文件名、大小和类型等。最后，利用 Response 的 WriteFile()方法将文件输出到客户端，再清除输出流。

例如，下面的代码演示了这一基本的下载过程：

```
//指定要下载文件的路径
string downFile = Server.MapPath("/software.rar");
//判断文件是否存在
if(File.Exists(downFile))
```

```
{
    //创建一个 fi 对象用于获取文件信息
    FileInfo fi = new FileInfo(downFile);
    //设置下载的默认文件名
    Response.AddHeader(
      "Content-Disposition", "attachment;filename=" + downFile);
    //设置下载的文件大小
    Response.AddHeader("Content-length", fi.Length.ToString());
    //设置下载的文件类型
    Response.ContentType = "application/octet-stream";
    //输出到客户端进行下载
    Response.WriteFile(fi.FullName);
    //清除输出流
    Response.End();
}
```

11.7.2 实例描述

还是书店网站的那点事儿，简直是没完没了。口水战之后，决定再添加最后一个功能，就是实现文件的下载。当然不是任何位置的任何文件都可以下载，必须以他们网站的 resource 作为根目录。将此目录下的文件给列出来，再选择下载哪一个文件。

说得简单，做起来难。对应到程序中，还得设计流程、编码实现、测试、处理异常，一步也不能少。

作为这个项目的终结功能，下面是作者的制作步骤。

11.7.3 实例应用

【例 11-7】文件下载。

(1) 在书店网站中新建一个页面，再添加一个 Panel 控件到页面内，该控件表示下载列表的显示位置：

```
<asp:Panel ID="Panel1" runat="server"></asp:Panel>
```

(2) 固定好了位置，这一步进入页面的.cs 文件。在页面加载完成后，获取 resource 目录下的所有文件，并绑定到前台的 Panel 控件中：

```
string root;
Table tableDirInfo = new Table();
protected void Page_Load(object sender, EventArgs e)
{
    root = Server.MapPath("/resource");        //指定下载文件的根目录
    GetFileList(root);                        //调用 GetFileList()方法获取文件列表
    Panel1.Controls.Add(tableDirInfo);        //将文件列表添加到指定位置
}
```

(3) 这一步编写上面所需的 GetFileList()方法，它有一个参数用于指定要显示文件列表的目录，在本例中为网站下的 resource 目录。经过一番调整，最终它的代码如下：

```
private void GetFileList(string strCurDir)
{
    string FileName, FileExt;
    long FileSize;                          //文件大小
    DateTime FileModify;            //最后修改时间
    if (Directory.Exists(Path.GetDirectoryName(strCurDir))) //判断目录是否存在
    {
        FileInfo fi;
        DirectoryInfo dir;
        TableCell td;
        TableHeaderCell th;
        TableRow tr;
        tr = new TableRow();      //动态添加单元格内容
        th = new TableHeaderCell();
        th.Controls.Add(new LiteralControl("文件名"));
        tr.Cells.Add(th);
        th = new TableHeaderCell();
        th.Controls.Add(new LiteralControl("文件类型"));
        tr.Cells.Add(th);
        th = new TableHeaderCell();
        th.Controls.Add(new LiteralControl("文件大小"));
        tr.Cells.Add(th);
        th = new TableHeaderCell();
        th.Controls.Add(new LiteralControl("最后修改时间"));
        tr.Cells.Add(th);
        th = new TableHeaderCell();
        th.Controls.Add(new LiteralControl("操作"));
        tr.Cells.Add(th);
        tableDirInfo.Rows.Add(tr);
        //针对当前目录建立目录引用对象
        DirectoryInfo dirInfo = new DirectoryInfo(strCurDir);
        //循环判断当前目录下的文件和目录
        foreach (FileSystemInfo fsi in dirInfo.GetFileSystemInfos())
        {
            FileName = "";
            FileExt = "";
            FileSize = 0;
            if (fsi is FileInfo)                //如果是文件
            {
                fi = (FileInfo)fsi;
                FileName = fi.Name;             //取得文件名
                FileExt = fi.Extension;         //取得文件的扩展名
                FileSize = fi.Length;           //取得文件的大小
                FileModify = fi.LastWriteTime;    //取得文件的最后修改时间
            }
            else                        //否则是目录
```

```
            {
                dir = (DirectoryInfo)fsi;
                FileName = dir.Name;            ///取得目录名
                FileModify = dir.LastWriteTime;   //取得目录的最后修改时间
                FileExt = "文件夹";              //设置文件的扩展名为"文件夹"
                //获取文件夹路径
                GetFileList(strCurDir + "\\" + dir.Name);
            }
            //动态添加表格内容
            tr = new TableRow();
            td = new TableCell();               //文件名称
            td.Controls.Add(new LiteralControl(FileName));
            tr.Cells.Add(td);
            td = new TableCell();               //文件类型
            td.Controls.Add(new LiteralControl(FileExt));
            tr.Cells.Add(td);
            td = new TableCell();               //文件大小
            td.Controls.Add(new LiteralControl(FileSize.ToString() + "字节"));
            tr.Cells.Add(td);
            td = new TableCell();               //最后修改时间
            td.Controls.Add(new
              LiteralControl(FileModify.ToString("yyyy-mm-dd hh:mm:ss")));
            tr.Cells.Add(td);
            td = new TableCell();               //文件的下载链接
            td.Controls.Add(new LiteralControl("<a href=?dFile="
              + Server.UrlEncode(fsi.Name) + ">下载</a>"));
            tr.Cells.Add(td);
            tableDirInfo.Rows.Add(tr);
        }
    }
}
```

代码有点长，主要是它做的事情比较多。首先判断传入的目录是否存在，如果存在，就创建一个 5 列的表格准备存放文件，包括文件名、文件类型、文件大小、最后修改时间和操作。

接下来，则开始做正事，遍历目录下的每个文件和目录，按照各列获取相应的信息，并添加到表格中。这里要注意最后一列中添加的文件下载链接，其中对文件名进行了编码转换。

(4) 注意“下载”链接指向了本页，并在 URL 中用 dFile 参数指定要下载文件的名称。对 Page_Load()中的代码进行补充，增加对下载文件的判断并处理下载事件。代码如下：

```
if (!IsPostBack)
{
    //是否有文件要下载
    if (Request.QueryString["dFile"]!= null
      && Request.QueryString["dFile"]!="")
    {
        //要下载文件的名称
        string downFile = Request.QueryString["dFile"].ToString();
        //调用 DownLoadFile()方法下载该文件
```

```
        DownLoadFile(downFile);
    }
}
```

(5) 真正用来下载的代码并不多，它在上一步中被调用，由 DownLoadFile()方法实现。这里来完善它的内容，注意对文件名中乱码的处理：

```
//下载方法
private void DownLoadFile(string fileName)
{
    string filePath = Server.MapPath("/resource/") + fileName;
    if (File.Exists(filePath))
    {
        FileInfo file = new FileInfo(filePath);

        Response.ContentEncoding =
          System.Text.Encoding.GetEncoding("UTF-8"); //解决中文乱码
        Response.AddHeader("Content-Disposition", "attachment; filename="
          + Server.UrlEncode(file.Name));  //解决中文乱码

        Response.AddHeader("Content-length", file.Length.ToString());
        Response.ContentType = "appliction/octet-stream";
        Response.WriteFile(file.FullName);
        Response.End();
    }
}
```

11.7.4　运行结果

下面来看看我们的杰作吧。从书店网站中进入这里创建的页面，首先会看到一个表格形式的文件列表，里面的文件对应于网站下的 resource 目录。

我们把鼠标移到操作列的“下载”链接上，可以在下面的地址栏查看传递的参数信息。单击后，就能把它下载(保存)到本地，如图 11-9 所示。



图 11-9　文件下载的运行效果

11.7.5 实例分析


源码解析：

按照项目惯例，添加功能完成之后，编写总结文档。

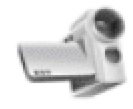
可以看出，下载服务器端文件所需的基本步骤如下。

先取得存放文件的目录地址，再取得该文件目录下所有的文件，接下来判断所取得文件的类型，然后将各个项的具体信息显示在一个表中。在点击下载时，其实就是对 HTTP 响应数据的操作，将文件内容显示在页面中。

最后，在对文件的下载过程中，需要注意添加判断文件是否存在的逻辑。

11.8 文件加密与解密

在安全领域中，加密与解密是一个永恒的话题。加密很明显，使明文变得不再可用，增强了安全性；解密则是一个逆过程，将乱码翻译为可直接使用的明文。最常见的加密算法有 DES、RSA 和 MD5。


视频教学：光盘/videos/11/Read&Write.avi　　　　长度：5 分钟

11.8.1 基础知识——按字节文件读写方法

ASP.NET 中提供了很多的系统类，能够完成对字符串的加密操作。但是，对于算法的控制不够灵活，因此有时候需要自己定义一套加密的算法。

例如，将原文与某个数进行移位或者异或操作产生密文，将原文存到一个字节数组再进行加密操作产生密文等。

1. ReadAllBytes()方法

位于 System.IO 命名空间中的 File 类提供了一个 ReadAllBytes()静态方法，可以打开一个文件，将文件的内容读入一个字符串，然后关闭该文件。

其定义如下：

```
public static byte[] ReadAllBytes(string path)
```

参数 path 表示要打开并进行读取的文件，返回值是一个包含文件内容的字节数组。

例如，下面的语句将“c:\windows\system.txt”文件的内容以字节数组的形式保存到 FileBytes 中：

```
byte[] FileBytes = File.ReadAllBytes(@"c:\windows\system.txt");
```

表 11-6 中列出使用 ReadAllBytes()方法读取文件时常见的需要处理的异常及说明。

表 11-6　读取文件时常见的异常

异常名称	说　明
ArgumentException	path 是一个零长度的字符串， 仅包含空白或者包含一个或多个由 InvalidPathChars 定义的无效字符
ArgumentNullException	path 为 null
PathTooLongException	指定的路径、文件名或者两者都超出了系统定义的最大长度。例如，在基于 Windows 的平台上，路径必须小于 248 个字符，文件名必须小于 260 个字符
DirectoryNotFoundException	指定的路径无效(例如，它位于未映射的驱动器上)
IOException	打开文件时发生了 I/O 错误
UnauthorizedAccessException	在当前平台上不支持此操作。path 指定了一个目录。调用方没有所要求的权限
FileNotFoundException	未找到 path 中指定的文件
NotSupportedException	path 的格式无效
SecurityException	调用方没有所要求的权限



ReadAllBytes()方法将根据现存的字节顺序标记来自动检测文件的编码，可检测到编码格式 GBK、UTF-8 和 UTF-32 等。

2. WriteAllBytes()方法

File 中还提供了对应 ReadAllBytes()方法的写入方法 WriteAllBytes()，它也是一个静态方法，用于创建一个新文件，在其中写入指定的字节数组，然后关闭该文件。

WriteAllBytes()方法的完整格式如下：

```
public static void WriteAllBytes(string path,  byte []bytes)
```

参数 path 表示要写入的文件，bytes 表示要写入文件的字节。该方法没有返回值，如果目标文件已存在，则改写该文件。

例如，下面的语句演示了如何使用 WriteAllBytes()方法将一个字符串写入到文件：

```
string str = "这些文本将用于WriteAllBytes()方法";         //指定字符串
byte []bytes = Encoding.UTF8.GetBytes(str.ToCharArray()); //转换为字节数组
File.WriteAllBytes(@"C:\test.txt", bytes);              //执行写入操作
```

11.8.2　实例描述

老板提供了一个国外公司自己软件生成的文件，让我们 decode 以后存成文本文件。先是直接打开，发现是一堆无法理解的乱码。进而用文件流形式读取，在 debug 模式下可以看到所有数据已被读出。但问题是一旦 output 到一个组件上面以后，所有在#0 之后的内容都没有显示出来。

另外，读出来的都是乱码的、像繁体中文之类的东西，怎么样转成 ASCII 码进而转成 String、Int 之类的呢？

一个解密文件，竟然出现这么多的问题。经过努力奋战，网上求助、自己实验，费了九牛二虎之力终于在规定的期限内实现了。

利用工作之余，准备把解决这个问题过程中使用的一些知识与技巧整理成文档，然后发布到博客上。这样，即做了笔记，方便以后查看，还可以帮助遇到类似问题的博友。

11.8.3 实例应用

【例 11-8】文件下载。

首先是搭建前台的布局，作者比较懒，直接拿以前用的项目后台界面进行修改，而且作为技术演示，也不需要多华丽的界面。如图 11-10 所示是最终设计的界面。

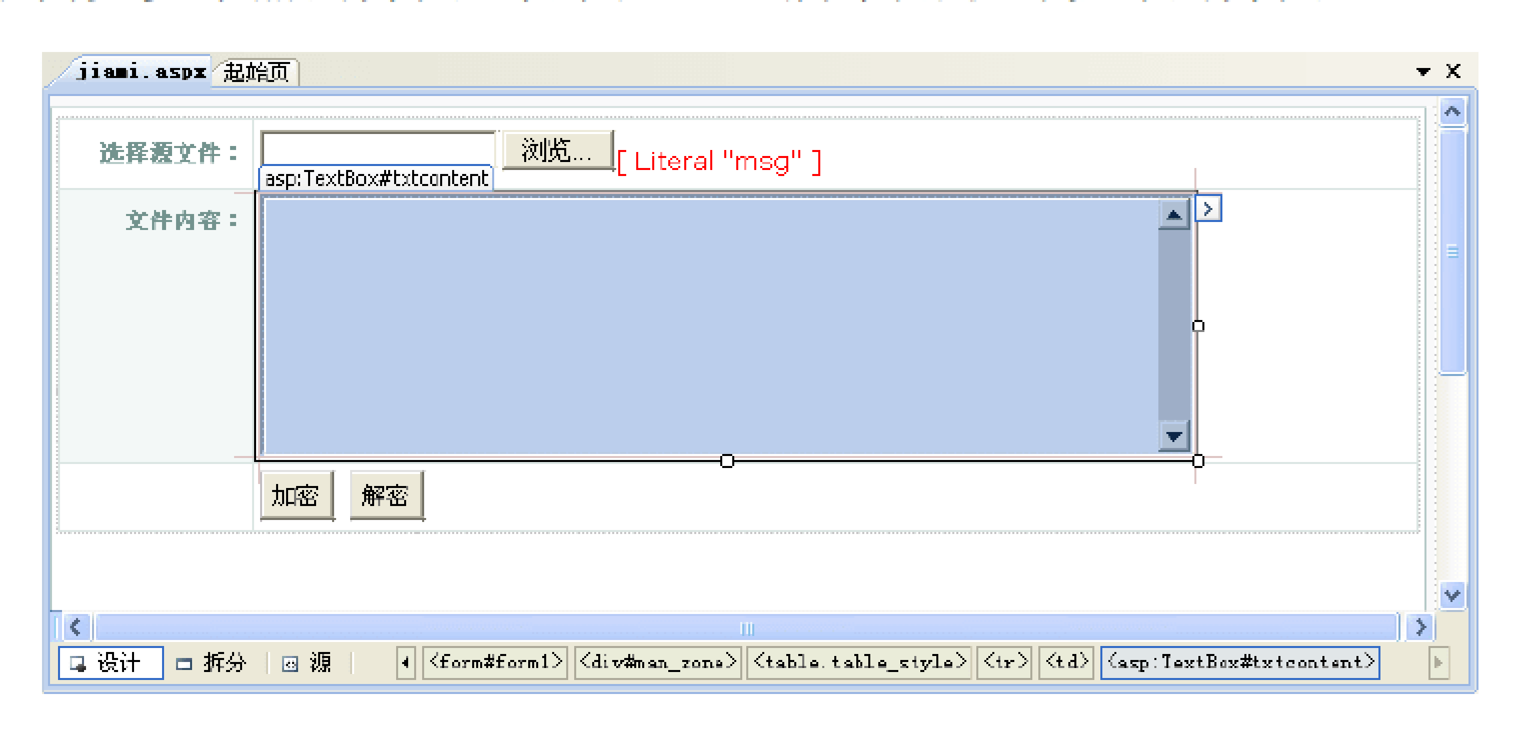


图 11-10 设计的界面

内行人士一看就知道，使用 FileUpload 控件来选择文件，Literal 控件显示提示信息，TextBox 控制则设置为多行显示文件内容，再提供了两个按钮用于选择要执行的功能。

下面我们来实现加密功能。这个功能实际上就是按照一定的顺序(规律)将文件的原始内容打乱，然后再保存：

```
//加密按钮
protected void btnmd_Click(object sender, EventArgs e)
{
    if (this.FileUpload1.PostedFile.ContentType
      != "text/plain")   //判断文件是否为空
    {
        msg.Text = "请选择一个文本文件！";
    }
    else
    {
        string filetype = FileUpload1.PostedFile.ContentType;   //获取文件类型
        string name = FileUpload1.PostedFile.FileName;  //获取文件名称
        byte []all = File.ReadAllBytes(name);           //按字节读取文件内容
        string []allline = new string[all.Length];    //创建与内容相同大小的数组
        for (int i=0; i<all.Length; i++)             //循环数组
```

```
        {
            allline[i] = Convert.ToString(all[i], 16);    //转换为十六进制字符串
        }
        File.WriteAllLines(name, allline);           //写入文件
        msg.Text = "加密成功！";                      //显示提示信息
        readFileContent(name);                   //载入文件并显示到 TextBox
    }
}
```

从上面的代码中可以看出，此处采用的加密方法为：将文件的内容全部读取到字节数组，再创建相同大小的字符串数组。然后，将字节数组中的每个元素按十六进制转换为字符串，再保存到字符串数组，最后将它写入文件。

解密的方法是一个逆加密过程，即按十六进制转换为字节，再按顺序写入文件。实现代码如下：

```
//解密按钮
protected void btnunmd_Click(object sender, EventArgs e)
{
    if (this.FileUpload1.PostedFile.ContentType
      != "text/plain")   //判断文件是否为空
    {
        msg.Text = "请选择一个文本文件！";
    }
    else
    {
        string name = FileUpload1.PostedFile.FileName;  //获取文件名称
        string []allLine = File.ReadAllLines(name); //按字符串读取文件内容
        byte []allStr = new byte[allLine.Length]; //创建与内容相同大小的字节数组
        for (int i=0; i<allLine.Length; i++)  //循环每个数组元素
        {
            int er = Convert.ToInt32(allLine[i], 16);    //将十六进制转为十进制
            allStr[i] = Convert.ToByte(er);             //再转为字节
        }
        File.WriteAllBytes(name, allStr);               //写入文件
        msg.Text = "解密成功！";                      //显示提示信息
        readFileContent(name);                   //载入文件并显示到 TextBox
    }
}
```

11.8.4　运行结果

运行文件，可以看到多么熟悉的后台界面。单击“浏览”按钮选择一个 TXT 文件作为测试，之后单击“加密”按钮，即可看到加密后的文件内容，如图 11-11 所示。

再次选择刚才的文件，如果单击“解密”按钮，将看到文件的明文，如图 11-12 所示。如图 11-13 所示为从记事本中查看原文件内容的效果。

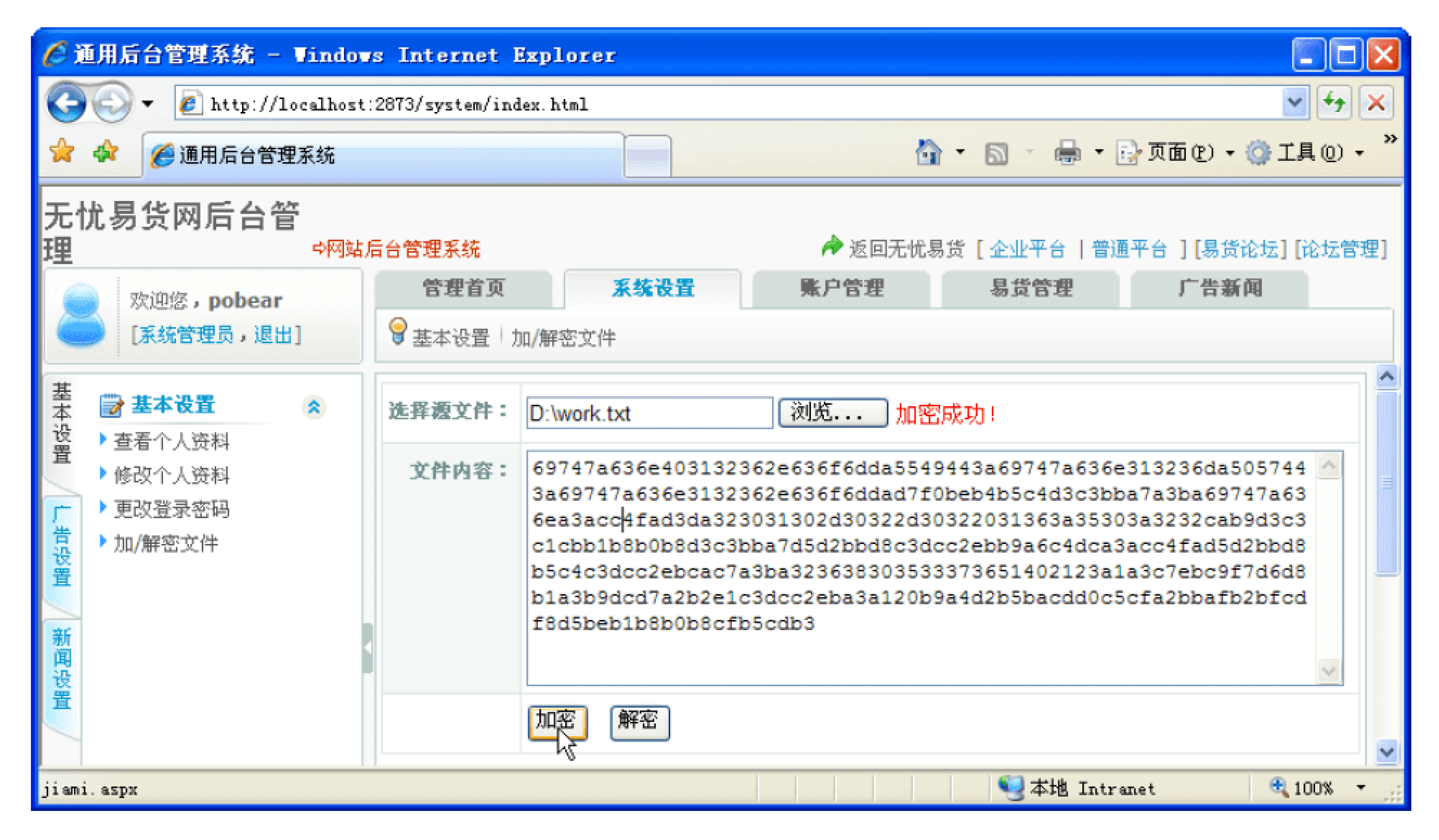


图 11-11　文件加密的效果



图 11-12　文件解密的效果

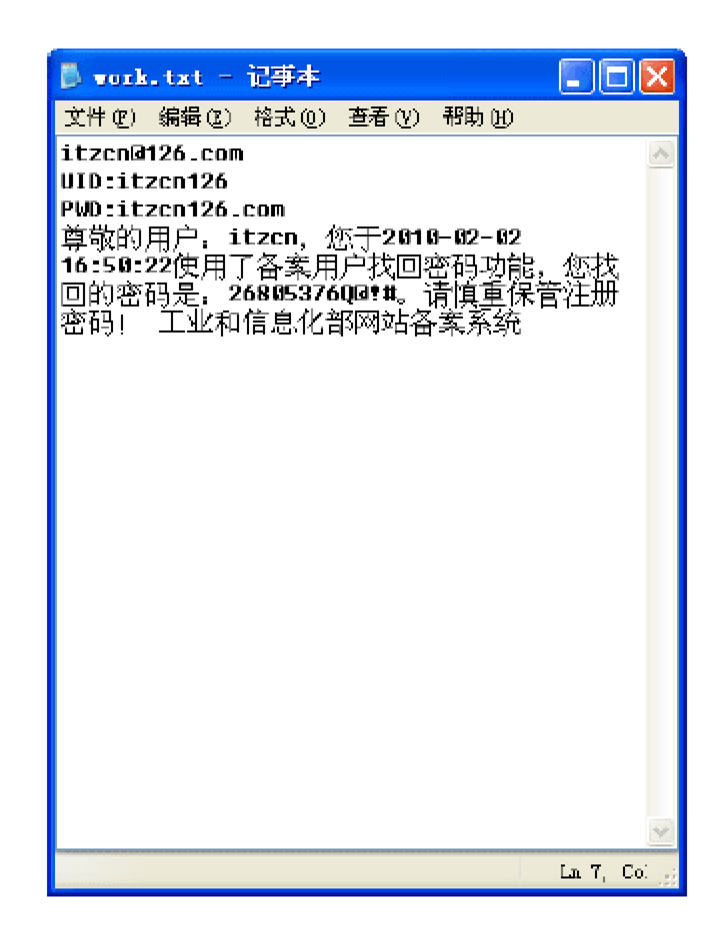


图 11-13　原文件内容

11.8.5　实例分析



源码解析：

思路为：将文本中的信息转换为字节，然后进行加密处理并写入文件。其中用到了 File 中的 ReadAllBytes()方法读取文件的字节数组，WriteAllBytes()方法写入文件的字节数组。

考虑到解密时的复杂性，这里仅仅对原文进行了进制转换。实际使用时可能要更复杂一些，如产生一个随机数进行位移或进行对称加密等。

11.9　删除网站下的非空目录

熟悉 Windows 的用户都知道，在选择一个目录或文件后，按 Delete 键可以直接将其删除，非常方便快捷。而对于一个网站来说，通常我们是用浏览器来查看文件，而且也无法像 Windows 那样方便地选择一个文件。因此，如何合理处理网站内文件的删除方式，成了很重要的话题。

 视频教学：光盘/videos/11/OperateDirectory.avi　　　　长度：11 分钟

11.9.1　基础知识——Delete()方法

ASP.NET 中，Directory 类的 Delete()方法可以实现删除指定的目录，有如下两种重载形式：

```
public static void Delete(string path);
public static void Delete(string path, bool recursive);
```

其中，第一个参数表示要删除的目录，当该目录不为空时，系统将会抛出异常，如图 11-14 所示。第二个参数为 bool 类型，为 true 时将会从目录中删除所有的子目录和文件。否则，会再次抛出 System.IO.IOException 异常。

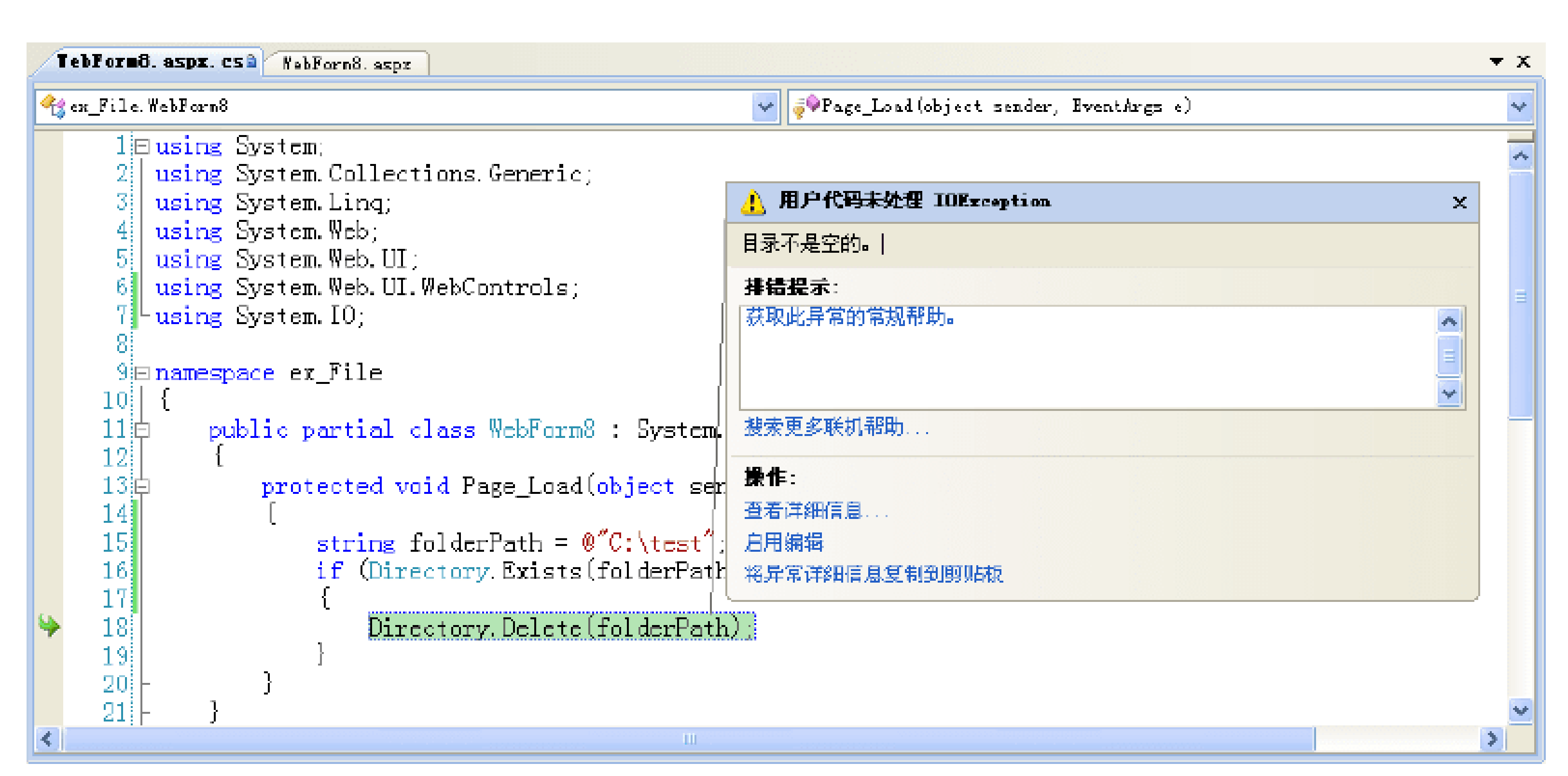


图 11-14　目录不为空的异常

例如，将“C:\test”目录及子目录删除：

```
string folderPath = @"C:\test";              //指定目录路径
if (Directory.Exists(folderPath))            //判断是否存在
{
    Directory.Delete(folderPath, true);      //删除该目录及子目录
}
```



用重载的 Delete 方法删除目录时，当前工作目录或者当前工作目录的子目录不能被删除。另外，如果一个程序引用了一个目录或者其中的文件，该目录也不能被删除。

表 11-7 列出了这些方法所能引起的异常类型及原因。

表 11-7 Directory类引起的异常类型及原因

异常类型	原　因
UnauthorizedAccessException	调用方没有所要求的权限
ArgumentException	参数是一个零长度字符串，仅包含空白。或者包含一个或多个无效字符
ArgumentNullException	参数为空引用
PathTooLongException	参数超出了系统定义的最大长度
IOException	参数是一个文件名
DirectoryNotFoundException	参数无效
SecurityException	调用方没有访问未委托的代码所需的权限
FileNotFoundException	未找到参数所指定的目录
NotSupportedException	参数的格式无效

11.9.2 实例描述

今天本来是周末，本想好好睡上它一天的，谁知老同学小祺又打来电话(这段时间让她给弄得都有点不敢接她电话了)。

通过了解才知道她又想加个删除文件功能(天啊！我那有且只有一天的星期天啊)。经过死磨硬缠我投降了，咱不多说了，赶紧结束休息日吧。

11.9.3 实例应用

【例 11-9】删除网站下的非空目录。

简单创建一个 ASPX 页面，在里面进行编辑。先用 Table1 来制作一个文件列表，指定各列的名称：

```
<asp:Table ID="Table1" runat="server" Width="99%" border="0" align="center"
  CellPadding="0" CellSpacing="1" bgcolor="#CECECE" onMouseOver="changeto()"
  onmouseout="changeback()">
    <asp:TableRow bgcolor="#FFFFFF" CssClass="thc">
        <asp:TableHeaderCell>文件名称</asp:TableHeaderCell>
        <asp:TableHeaderCell>文件类型</asp:TableHeaderCell>
        <asp:TableHeaderCell>文件大小</asp:TableHeaderCell>
        <asp:TableHeaderCell>最后修改时间</asp:TableHeaderCell>
        <asp:TableHeaderCell>操作</asp:TableHeaderCell>
    </asp:TableRow>
</asp:Table>
```

表格很简单，为了使显示效果不是很丑陋，直接套用了以前的 CSS 样式和 JavaScript 代码。原则是：能复用决不二次开发，包括模块也是。

创建 GetFileList()方法实现在 Table1 中显示文件：

```
private void GetFileList(string downpath)
{
    string FileName, FileExt;
    long FileSize;
    DateTime lasttime;
    if (Directory.Exists(downpath))
    {
        FileInfo fi;
        DirectoryInfo dir;
        DirectoryInfo dirInfo = new DirectoryInfo(downpath);
        foreach (FileSystemInfo fsi in dirInfo.GetFileSystemInfos())
        {
            FileName = "";
            FileExt = "";
            FileSize = 0;
            if (fsi is FileInfo)
            {
                fi = (FileInfo)fsi;
                FileName = fi.Name;
                FileExt = fi.Extension;
                FileSize = fi.Length;
                lasttime = fi.LastWriteTime;
            }
            else
            {
                dir = (DirectoryInfo)fsi;
                FileName = dir.Name;
                lasttime = dir.LastWriteTime;
                FileExt = "文件夹";
            }
            TableCell td;
            TableRow tr = new TableRow();
            tr.CssClass = "tc";
            td = new TableCell();
            td.Controls.Add(new LiteralControl(FileName));
            tr.Cells.Add(td);
            td = new TableCell();
            td.Controls.Add(new LiteralControl(FileExt));
            tr.Cells.Add(td);
            td = new TableCell();
            td.Controls.Add(new LiteralControl(FileSize.ToString() + "字节"));
            tr.Cells.Add(td);
            td = new TableCell();
            td.Controls.Add(
              new LiteralControl(lasttime.ToString("yyyy-MM-dd hh:mm:ss")));
            tr.Cells.Add(td);
            td = new TableCell();
```

```
            td.Controls.Add(new LiteralControl("<img src=\"images/010.gif\"
width=\"9\" height=\"9\" /> <a href='#' onclick=\"yesToDel('"
              + Server.UrlEncode(fsi.FullName) + "')\">删除</a>"));
            tr.Cells.Add(td);
            Table1.Rows.Add(tr);
        }
    }
}
```

代码好像有点长了。这段代码的作用是把网站下指定文件夹的所有文件以列表的形式显示出来，并控制显示出的格式和输出删除链接。

GetFileList()能够读取任何网站下的目录，在这里是作者网站下的 resource 目录，页面加载后即显示：

```
protected void Page_Load(object sender, EventArgs e)
{
    string downpath = Server.MapPath("~/resource");
    GetFileList(downpath); //调用获取文件的方法
}
```

接下来，我们处理“删除”链接被单击之后要做的事情。同样，从复用的角度出发，这里提炼出一个单独的方法来完成这件任务：

```
if (Request.QueryString["deFile"] != null
  && Request.QueryString["deFile"] != "")
{
    string downFile = Request.QueryString["deFile"].ToString();
    DeleteFile(downFile); //调用删除文件的方法
}
```

下面隆重介绍一下本实例的幕后主角——DeleteFile()方法，它判断要删除的是文件还是目录，再执行删除操作，之后告诉我们成功的消息：

```
private void DeleteFile(string downFile)
{
    if (File.Exists(downFile))
    {
        File.Delete(downFile);
        JavaScriptHelper.Alert(
          "删除" + downFile.Replace("\\", "\\\\") + "成功.", "tab.aspx");
    }
    else
    {
        if (Directory.Exists(downFile))
        {
            Directory.Delete(downFile, true);
            JavaScriptHelper.Alert(
              "删除" + downFile.Replace("\\", "\\\\") + "成功.", "tab.aspx");
        }
        else
```

```
            {
                JavaScriptHelper.Alert("指定的不是有效文件或目录!");
            }
        }
}
```

整个结构和流程很容易理解，而且它还对出错的情况进行了处理，是多么人性化啊。好了，至此实例的主体框架都已经搭建好。

最后，我们来编写提示信息的对话框，将它们封装到名为 JavaScriptHelper 的类中：

```
public static void Alert(string message)
{
    string js = @"<Script language='JavaScript'>alert('"
      + HttpContext.Current.Server.UrlDecode(message) + "');</Script>";
    HttpContext.Current.Response.Write(js);
}
public static void Alert(string message, string url)
{
    string js = @"<Script language='JavaScript'>"
      + " alert('" + HttpContext.Current.Server.UrlDecode(message) + "');"
      + "top.location.href='" + url + "';</Script>";
    HttpContext.Current.Response.Write(js);
}
```

有兴趣的读者可以在这里添加更多 JavaScript 代码，实例中仅用到了直接弹出对话框，以及弹出对话框后转向一个 URL 地址。

11.9.4　运行结果

全部搞定了，开始运行查看最终效果。看看设计的界面以及文件列表，效果很不错哦。

先测试一下目录的删除效果，当我们单击目录后面的“删除”链接时给出提示，询问用户是否确认以免误删除。当选择“取消”时，同样会给出反馈信息，如图 11-15 所示。

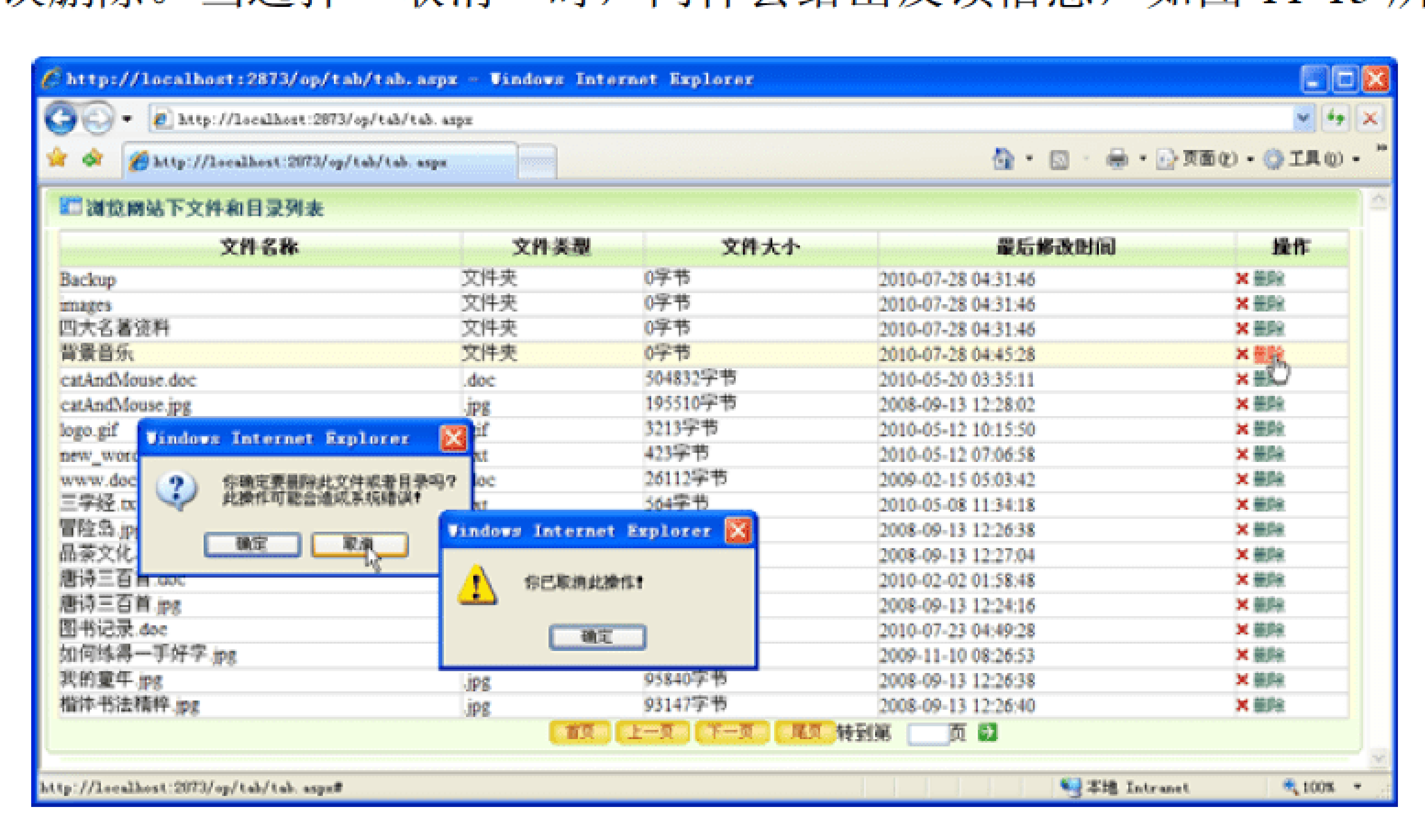


图 11-15　取消删除目录的效果

重新开始，这次在询问对话框中单击“确定”按钮，此时会将要删除的目录信息提交到服务器处理。完成后会看到删除成功对话框，这从图 11-16 中可以看出。

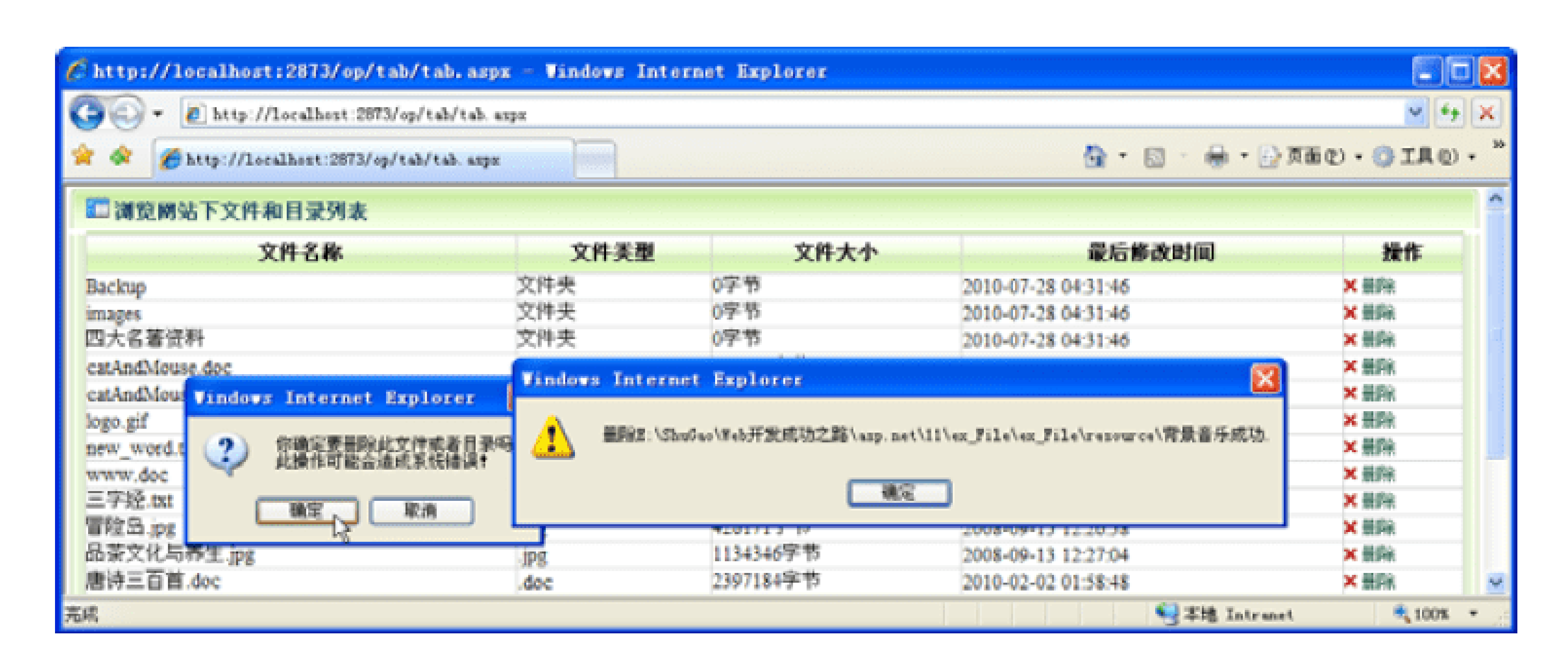


图 11-16 删除目录成功的效果

目录测试到此结束吧，下面试试删除一个文件。又看到了熟悉的询问对话框，如图 11-17 所示，很确定地要删除这个文件，之后看到成功提示，如图 11-18 所示。

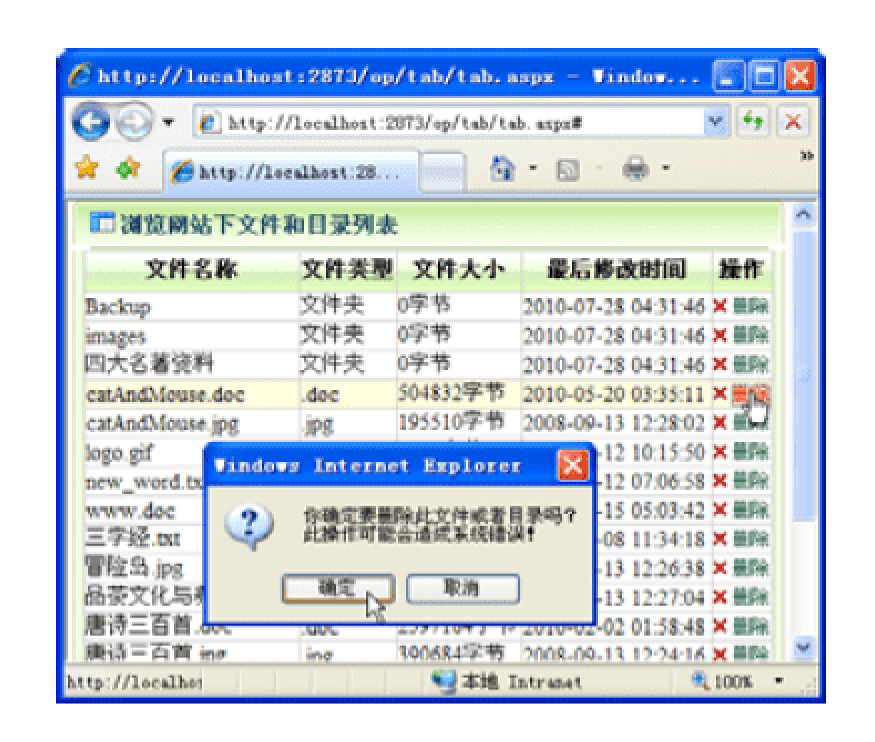


图 11-17 删除文件提示对话框

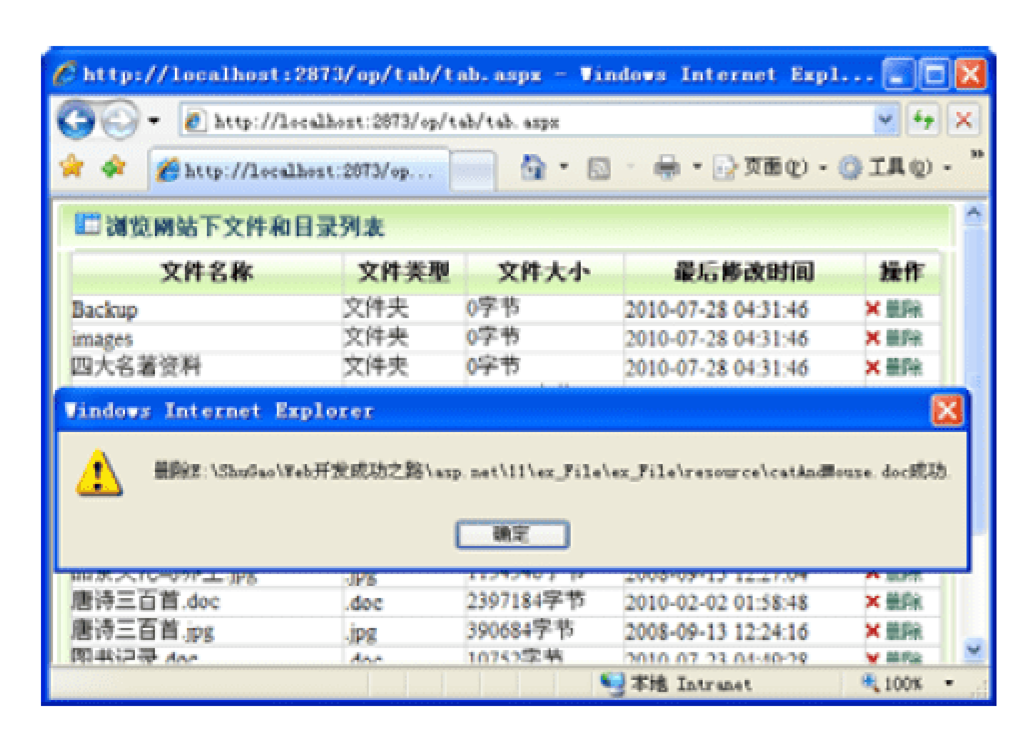


图 11-18 删除文件成功的效果

11.9.5 实例分析



源码解析：

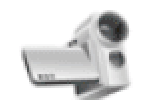
从本实例中可以看出，删除是对下载文件的扩充，对普通文件只是调用 File.Delete()方法。而对文件夹类型文件调用 Directory.Delete()方法。

删除时需要判断文件是否存在，在删除过程中需要用户确认是否删除文件，为了删除后页面所显示的文件列表是最新的，将显示文件；列表的方法放在了删除方法之后，即 GetFileList()方法。其中删除文件时如果删除的是文件夹，还将提示是否删除此文件夹的全部内容。

11.10 检测系统安装路径

运行于 Web 服务器上的 ASP.NET 网站的性能，在很大程度上取决于所在的操作系统。特

别是对于一些第三方的组件，由于它们不是 ASP.NET 中内置的，因此在使用时必须检测系统路径下是否有该组件，否则容易造成程序异常或者崩溃。



视频教学：光盘/videos/11/System.avi　　长度：5 分钟

11.10.1 基础知识——Environment类

使用 System.Environment 类可检索各种客户端操作系统的信息，如命令行参数、退出代码、环境变量设置、自上次系统启动以来的时间，以及公共语言运行时的版本等。

在表 11-8 中列出了 Environment 类中与系统相关的常用属性及说明。

表 11-8 常用的系统相关属性

属性名称	说 明
CurrentDirectory	获取或设置当前工作目录的完全限定路径
DeviceType	获取一个值，该值标识设备类型
ExitCode	获取或设置进程的退出代码
HasShutdownStarted	获取一个值，该值指示是否公共语言运行时正在关闭或者当前的应用程序域正在卸载
NewLine	获取为此环境定义的换行字符串
OSVersion	获取包含当前平台标识符和版本号的 OperatingSystem 对象
ProcessorCount	获取当前计算机上的处理器数
TickCount	获取系统启动后经过的毫秒数
Version	获取一个 Version 对象，该对象描述公共语言运行时的主版本、次版本、内部版本和修订号
SystemDirectory	获取系统目录的完全路径
MachineName	获取本地计算机的 NetBIOS 名称

在方法中最常用的是 GetFolderPath()方法，它可以获取指向指定枚举标识的系统特殊文件夹的路径。

语法格式如下：

```
public static string GetFolderPath(
    Environment.SpecialFolder folder
)
```

参数：folder 是用于标识系统的特殊文件夹，为 System.Environment.SpecialFolder 枚举中的任意常数，如表 11-9 所示。

返回值：返回一个字符串。如果指定系统的特殊文件夹实际存在于计算机上，则返回该文件夹的路径；否则为空字符串。



如果系统未创建文件夹、已删除现有文件夹，或者文件夹是不对应物理路径的虚拟目录(例如“我的电脑”)，则该文件夹不会实际存在。

表 11-9　SpecialFolder枚举常数

成员名称	说　明
ApplicationData	目录，用作当前漫游用户的应用程序特定数据的公共储存库
CommonApplicationData	目录，用作所有用户使用的应用程序特定数据的公共储存库
LocalApplicationData	目录，用作当前非漫游用户使用的应用程序特定数据的公共储存库
Cookies	用作 Internet Cookie 的公共储存库的目录
Desktop	逻辑桌面，而不是物理文件系统位置
Favorites	用作用户收藏夹项的公共储存库的目录
History	用作 Internet 历史记录项的公共储存库的目录
InternetCache	用作 Internet 临时文件的公共储存库的目录
Programs	包含用户程序组的目录
MyComputer	“我的电脑”文件夹
MyMusic	My Music 文件夹
MyPictures	My Pictures 文件夹
MyVideos	My Videos 文件夹
Recent	包含用户最近使用过的文档的目录
SendTo	包含“发送”菜单项的目录
StartMenu	包含“开始”菜单项的目录
Startup	对应于用户的“启动”程序组的目录
System	System 目录
Templates	用作文档模板的公共储存库的目录
MyDocuments	“我的文档”文件夹
ProgramFiles	Program files 目录
CommonProgramFiles	用于应用程序间共享的组件的目录

例如，下面的示例演示如何使用 Environment 类显示有关当前环境的信息列表：

```
static void Main(string []args)
{
    Console.WriteLine("-- 使用 Environment 类 --" );
    Console.WriteLine("ExitCode: {0}", Environment.ExitCode);
    Console.WriteLine(
      "HasShutdownStarted: {0}", Environment.HasShutdownStarted);
    Console.WriteLine("NewLine: {0} first line{0} second line{0} third line",
      Environment.NewLine);
    Console.WriteLine("OSVersion: {0}", Environment.OSVersion.ToString());
    Console.WriteLine("TickCount: {0}", Environment.TickCount);
    Console.WriteLine("Version: {0}", Environment.Version.ToString());
    Console.WriteLine("Desktop: {0}",
      Environment.GetFolderPath(Environment.SpecialFolder.Desktop));
    Console.ReadKey();
}
```

运行后的效果如图 11-19 所示。

```
file:///E:/ShuGao/Web开发成功之路/asp.net/11/ex_File/ConsoleApplication...
-- 使用Environment类 --
ExitCode: 0
HasShutdownStarted: False
NewLine:
  first line
  second line
  third line
OSVersion: Microsoft Windows NT 5.1.2600 Service Pack 3
TickCount: 2517984
Version: 2.0.50727.3603
Desktop: C:\Documents and Settings\Administrator\桌面
```

图 11-19　显示当前环境的信息列表

11.10.2　实例描述

如何获得本地机器的名称？

相信做网站开发的读者都遇到过类似的问题。作者就曾在一个项目中为此功能而苦战了两天还没弄出正确结果。后来有幸得到一位专家的妙方才得以实现(对方是微软全球技术中心 ASP.NET 技术支持顾问)。

下面给出解决与实现方法，与有兴趣的读者共同来研究。

11.10.3　实例应用

【例 11-10】检测系统安装路径。

首先不得不承认 ASP.NET 功能的强大，最大的特点就是提供了功能丰富的类库，从基础数据类型→对象与委托机制→数据库控制，再到操作页面→操作客户端→服务器端控制等。这些类库极大地提高了开发人员的工作效率和代码质量。

实现上面的功能其实就是利用了类库中一个类。看来平时学习和所用的都是冰山一角啊。

转入正题，在网站内创建一个 ASPX 页面。然后设置在哪个位置显示结果，这里采用表格+Literal 控件形式，每行一条信息。

如图 11-20 所示为布局后的效果。

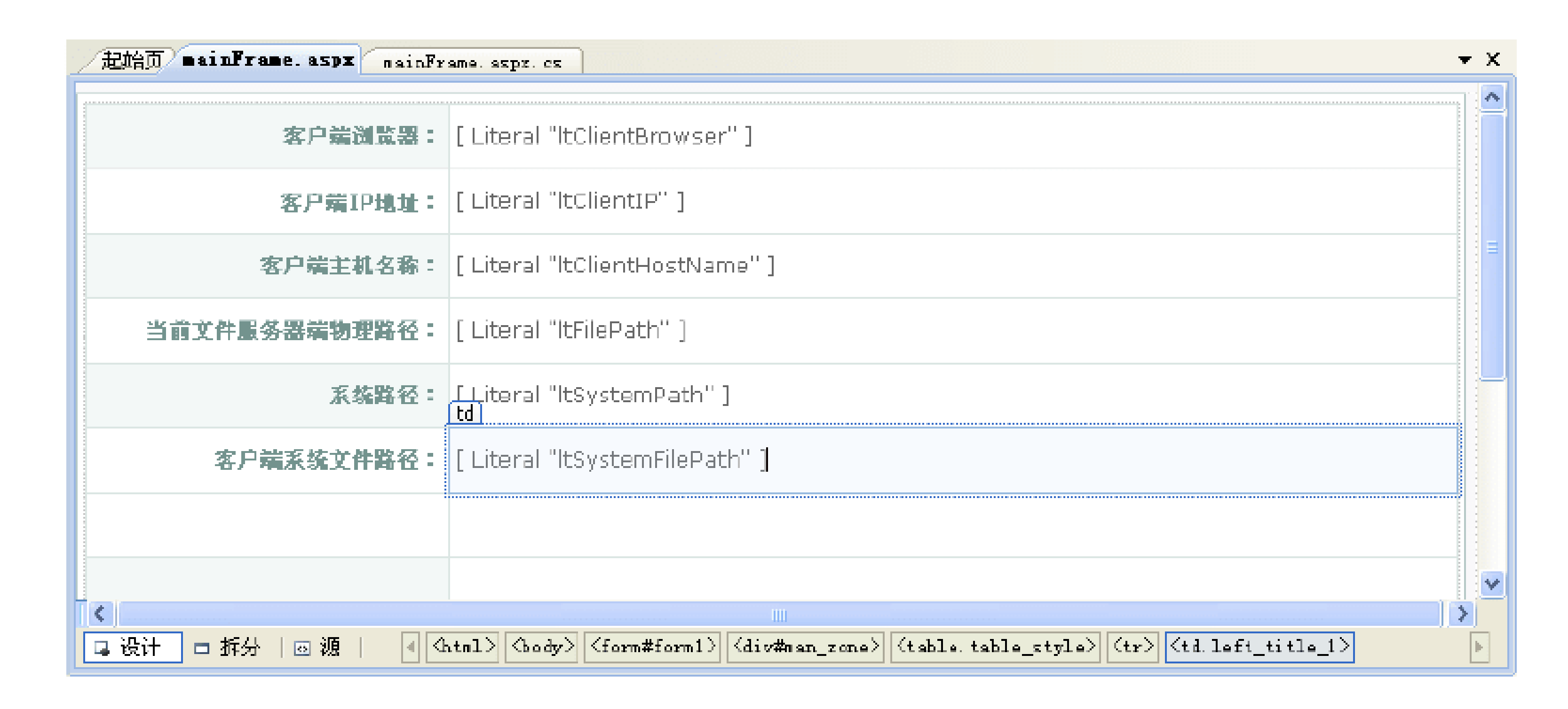


图 11-20　制作布局效果

接下来转到后台，控制 Literal 控件显示的文本，使用了 Request 对象和 Environment 类：

```
protected void Page_Load(object sender, EventArgs e)
{
    //客户端浏览器
    ltClientBrowser.Text = Request.UserAgent;
    //客户端 IP 地址
    ltClientIP.Text = Request.UserHostAddress;
    //客户端主机名称
    ltClientHostName.Text = System.Environment.MachineName;
    //当前文件服务器端物理路径
    ltFilePath.Text = Request.PhysicalApplicationPath;
    //系统路径
    ltSystemPath.Text = Path.GetDirectoryName(Environment
      .GetFolderPath(Environment.SpecialFolder.System));
    //客户端系统文件路径
    ltSystemFilePath.Text = System.Environment.SystemDirectory;
}
```

11.10.4　运行结果

Environment 类就是那位专家介绍给我新认识的，赶紧运行看看它是否能正常工作吧。效果如图 11-21 所示，正确地显示了客户端主机名称和系统安装路径等信息。



图 11-21　查看运行效果

11.10.5　实例分析



源码解析：

运行后出现客户端的各种信息，其中使用了 System.Environment 类获取主机名称、系统路

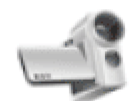
径和系统文件路径。

Environment 类虽然不常用，但它几乎能够提供所有有关当前环境和平台的信息，以及操作它们的方法。可惜的是，我们无法继承此类，否则便可以轻易地对客户端进行控制了。

11.11　简易文件浏览器

文件浏览器一个非常实用的功能，在 Windows 中它可以对文件和目录进行查看、新建、修改和删除等操作，在文件上传中可以选择文件保存的目录，还有在浏览网站的文件时需要指定所在的目录以及文件名称。

本节将创建一个简易的文件浏览器，使我们能够很好地掌握管理文件和目录的各种知识。

 **视频教学：**光盘/videos/11/OperateFile.avi　　长度：7 分钟

11.11.1　基础知识——File类

System.IO 命名空间的 File 类操作方式与 Directory 类相似，大多都为静态方法，这些静态方法可用来创建、删除、移动和打开文件。其中最常用的方法如下。

(1) Copy()

可用来把源文件复制到目标文件。可以使用绝对目录和相对目录引用。如果目标文件已经存在，或者目标文件是目录，将会产生异常。由 CLR 产生的异常随具体情况的不同而变化。

(2) Create()

可创建一个新的文件，返回 FileStream 类实例。如果文件已经存在，将会被重写而不会产生异常。因此，要想防止文件被意外重写，可以首先调用 Exists 方法来确保文件不存在。

(3) Delete()

用来删除文件。可以使用绝对目录和相对目录引用。如果目录或文件未找到，或者该方法的参数包含空字符串，CLR 将会产生异常。

(4) Exists()

接受一个文件名参数。它返回 bool 值指示文件是否存在。

(5) Move()

能把源文件移动到目标文件。可以使用该方法重命名文件，方法是使用不同的文件名，而源文件和目标文件的目录相同。该方法允许把文件从一个逻辑驱动器移动到另一个。

(6) GetAttributes()和 SetAttributes()

分别用于获取和设置文件属性，与 FileInfo 类的 Attributes 属性作用相同。可以定义文件的特征，如是否为隐藏文件、只读文件等。

(7) GetCreationTime()、GetLastAccessTime()和 GetLastWriteTime()

分别用于获取创建文件的日期和时间、最近一次访问文件的时间，以及最近一次写入文件的时间。

例如，下面的代码演示了如何使用 File 类创建、复制、移动和删除文件等操作：

```
File.Create("E:\\C#\\file.txt");                          //创建文件
File.Copy("E:\\C#\\file.txt","E:\\C#\\fileinfo.txt");        //复制文件
File.Move("E:\\C#\\file.txt","E:\\C#\\File\\file.txt");          //移动文件
File.Move("E:\\C#\\file.txt","E:\\C#\\File.txt");       //重命名文件
File.Delete("file.txt");                          //删除文件
```

第一行代码使用绝对目录引用在“E:\C#”中创建文件 file.txt。第二行、第三行和第四行代码分别使用绝对目录引用复制、移动和重命名文件。最后一行代码从当前工作目录中删除文件 file.txt。

技术文档	FileInfo类和File类的区别
FileInfo 类方法的使用与 File 类的方法使用很相似，并且有很多功能也相同。如 Copyto 方法与 File 类中的 Copy 方法就是相同的，不同的是 FileInfo 类中的方法不是静态的，必须通过 FileInfo 实例化对象进行访问，而 File 类就不必了。 如果用户打算多次重用某个对象，可以考虑使用 FileInfo 的实例方法，而不是 File 类的相应静态方法，因为 FileInfo 的实例方法并不总是需要进行安全检查。	

11.11.2 实例描述

OA(办公自动化)系统的一个模块要求制作一个文件浏览器。

具体为：在程序中能够将指定目录设为根目录，列出所有该目录下的文件和文件夹，单击某文件夹，列出该文夹下的所有文件夹和文件。可打开的文件或文件夹层次无限制。还要可以显示当前位置。

11.11.3 实例应用

【例 11-11】简易文件浏览器。

(1) 在 ASPX 页面中简单装修一下，设置背景、添加图片和分隔线什么的。再制作一个表头，包含文件名称、类型、大小和最后访问时间 4 列。

(2) 在表格下方插入一个 Repeater 控件，编辑 ItemTemplate 模板内的代码为如下所示，对应上面表头中的各列：

```
<asp:Repeater ID="Repeater1" runat="server">
   <ItemTemplate>
      <tr align="left">
      <td height="26"><img src='images/072.gif' width='9' height='8' />
      <asp:LinkButton ID="LinkButton1"
        runat="server" CommandArgument='<%# Eval("Name") %>'
        OnCommand="LinkButton1_Command">
      <%# Eval("Name") %>
      </asp:LinkButton>
      </td>
```

```
        <td width="187">
        <%# bool.Parse(Eval("IsFolder")+"")?"文件夹":"文件" %>
        </td>
        <td width="81"><%# Eval("Size")+"字节" %></td>
        <td width="192"><%# Eval("LastTime") %></td>
        </tr>
        <tr>
        <td colspan='5' height='1' background='images/073.gif'></td>
        </tr>
    </ItemTemplate>
</asp:Repeater>
```

(3) 根据要求还要能够返回上一级目录，在这里添加一个 Button 控件(单击事件将在后面介绍)：

```
<asp:Button ID="Button1" runat="server" Text="返回上级目录"
  OnClick="Button1_Click" />
```

(4) 追加一个 Literal 控件到列表之外来显示当前位于哪个目录中。如图 11-22 所示为 4 步后的最终布局效果。



图 11-22　制作文件浏览器布局

(5) 切换到后台的页面加载事件，修改代码为如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        string rootPath = Server.MapPath("~/resource");
        Session["RootPath"] = rootPath;
        ViewState["CurrentPath"] = "/";
        this.BindFileList("");  //绑定根文件目录
    }
    Literal1.Text = ViewState["CurrentPath"].ToString();
}
```

(6) 从列表中单击某个目录后，可查看该目录下的所有文件和子目录。每个列表项是一个 LinkButton 控件，编写 Command 事件代码：

```
protected void LinkButton1_Command(object sender, CommandEventArgs e)
```

```
{
    /* 进入下级文件夹   */
    string currentPath = ViewState["CurrentPath"] + "";
    string subPath = e.CommandArgument + "";
    string newPath = currentPath + subPath + "/";
    if (this.BindFileList(newPath)) //绑定子文件夹
    {
        ViewState["CurrentPath"] = newPath;
        Literal1.Text = ViewState["CurrentPath"].ToString();
    }
}
```

(7) “返回上级目录”是一个普通的按钮，单击后返回当前目录的父目录中。这一步编写该按钮的单击事件处理代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    /*返回上级文件夹*/
    string currentPath = ViewState["CurrentPath"] + "";
    string newPath = string.Empty;
    //查找从后往前倒数第二条斜线的位置
    int linePlace = this.GetSecondLinePlaceOnRight(currentPath);
    //如果位置大于 0，就截取，否则说明已到达根目录
    if (linePlace > 0)
    {
        newPath = currentPath.Substring(0, linePlace);
    }
    else
    {
        newPath = "";
    }
    this.BindFileList(newPath); //绑定子文件夹
    ViewState["CurrentPath"] = newPath;
    Literal1.Text = ViewState["CurrentPath"].ToString();
}
```

(8) 编写上步中调用的 GetSecondLinePlaceOnRight()方法，该方法从保存的当前目录中截取出父目录：

```
/// <summary>
/// 获取路径右边第二条斜线的位置
/// </summary>
private int GetSecondLinePlaceOnRight(string path)
{
    if (path == string.Empty) return 0;
    path = path.Substring(0, path.Length - 1);
    return path.LastIndexOf('/');
}
```

(9) 实现绑定文件信息到前台 Repeater 控件并显示的方法 BindFileList()，它同样在前面多

次被调用，带一个参数来指定要浏览的目录：

```
/// <summary>
/// 绑定文件列表
/// 在 Session 保存的 RootPath 基础上打开目录
/// 如果目录不存在，返回 false
/// </summary>
private bool BindFileList(string path)
{
    string root = Session["RootPath"] + "\\";
    DirectoryInfo dir = new DirectoryInfo(root + path);
    if (!dir.Exists) return false;  //如果目录不存在，返回 false
    IList<MyFileInfo> files = new List<MyFileInfo>();
    DirectoryInfo []dirs = dir.GetDirectories(); //获取当前目录下的所有子目录
    FileInfo []fils = dir.GetFiles();   //获取当前目录下的所有文件
    foreach (DirectoryInfo di in dirs)
    {
        files.Add(new MyFileInfo() {
            IsFolder = true,
            Name = di.Name,
            Size = "0",
            LastTime = di.LastAccessTime.ToString("yyyy-MM-dd hh:mm:ss")
        });
    }
    foreach (FileInfo fi in fils)
    {
        files.Add(new MyFileInfo() {
            IsFolder = false,
            Name = fi.Name,
            Size = fi.Length.ToString(),
            LastTime = fi.LastAccessTime.ToString("yyyy-MM-dd hh:mm:ss")
        });
    }
    //绑定文件列表
    Repeater1.DataSource = files;
    Repeater1.DataBind();
    return true;
}
```

(10) 为更方便地绑定对象，还需要声明一个用以封装文件信息的实体类 MyFileInfo。MyFileInfo 类的定义如下：

```
class MyFileInfo
{
    /* 文件实体
    包括文件名称、文件类型、文件大小、最后访问日期
    */
    public string Name { get;   set; }
    public bool IsFolder  { get;  set; }
    public string Size { get; set; }
```

```
    public string LastTime { get; set; }
}
```

(11) 经过前面 10 个步骤，这个简单的文件浏览器就实现了。

11.11.4　运行结果

运行 ASPX 页面后，程序会遍历指定目录下的所有文件和目录并绑定到 Repeater 控件中进行显示，而且还有“返回上级目录”按钮和显示当前目录。运行效果如图 11-23 所示。



图 11-23　默认运行后的效果

单击一个类型为“文件夹”的名称即可进入该文件夹，同时列出该文件夹下的所有文件夹和文件，如图 11-24 所示。最后单击“返回上级目录”按钮可以返回，直到根文件夹后便无效。

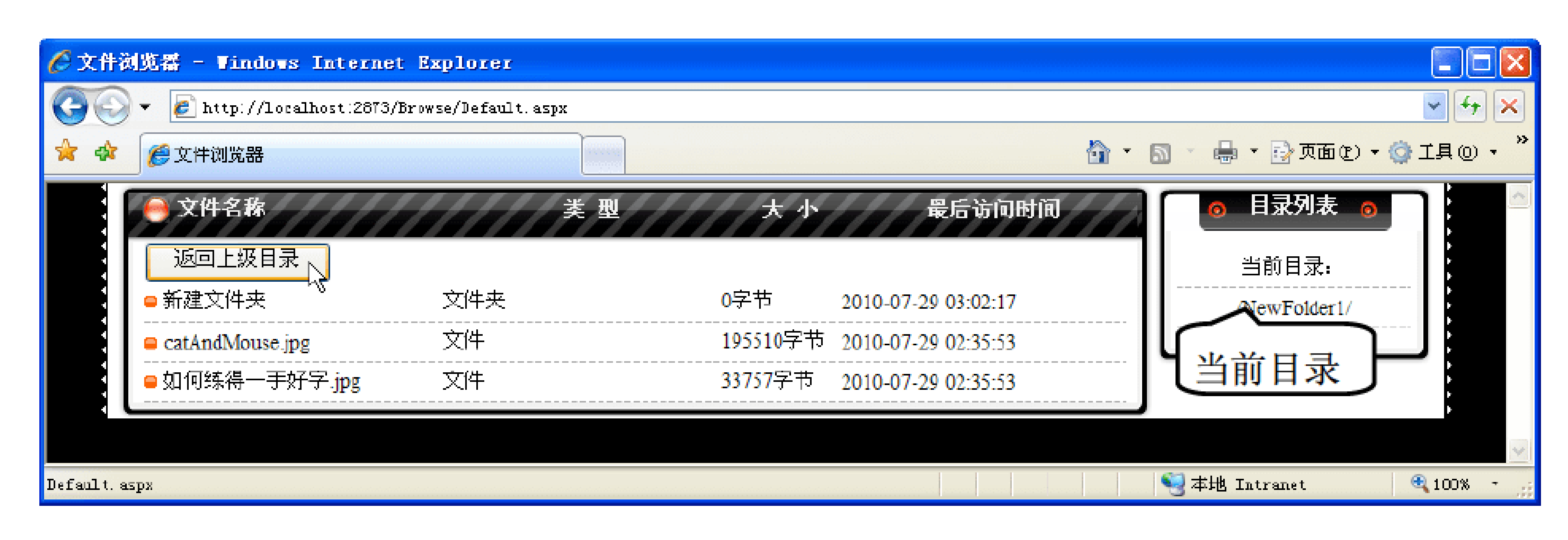


图 11-24　查看子目录的效果

11.11.5　实例分析



源码解析：

创建一个指定目录的 DirectoryInfo 类对象，遍历该目录下的所有子目录和所有文件。

本实例的细节关键在于目录的控制。本实例使用 ViewState[“CurrentPath”]保存当前目录的

字符串信息，每次目录操作对该字符串进行追加或截取。

为了更简单地展示在页面上，本实例使用了 Repeater 控件和 MyFileInfo 类辅助展示信息。

11.12　实现在线故事接龙游戏

所谓故事接龙，就是指可以让网友自由写故事，并且由网友提供故事情节发展路线，供其他网友接龙。这类程序大多数用的是论坛程序做的，但是也可以做成 txt 文件。这里打算以文件方式为例说一下如何实现。



视频教学： 光盘/videos/11/Read&Write.avi　　　　长度：5 分钟

11.12.1　基础知识——文件流读写方法

对文件的读写操作应该是最重要的文件操作，System.IO 命名空间为我们提供了很多有关文件的读写操作。本节将介绍最常用、也是最基本的 StreamReader 类和 StreamWriter 类，它们是基于流的读写操作类。

1. StreamReader类

StreamReader 类提供了读取文件的功能，它不仅可以读取文件，还可以处理任何流信息。该类为读取文件提供了很多方法，其中常用的有 Close()、Read()、ReadLine()、ReadToEnd()和 Peek()。

下面的代码演示了如何使用 StreamReader 类读取文件：

```
//指定要读取文件的路径
string fileName = Server.MapPath("/all.txt");
//保存内容的变量
string fileTexts = "";
//文件是否存在
if (File.Exists(fileName))
{
    //打开文件
    FileStream fileStream = File.OpenRead(fileName);
    try
    {
        //创建文件读取流
        StreamReader reader =
          new StreamReader(fileStream, System.Text.Encoding.Default);
        //是否读取完成
        while (!reader.EndOfStream)
        {
            //逐行读取并保存
            fileTexts += reader.ReadLine() + "\r\n";
        }
```

```
        //输出文件内容
        Response.Write("内容为: " + fileTexts);
        //关闭流
        reader.Close();
    }
    catch (Exception ex)
    {
        //输出错误信息
        Response.Write(ex.Message);
    }
}
```

在上述代码中实例化了一个 StreamReader 类 reader，而且指定了使用的字符集，从而避免了乱码的出现。然后通过 StreamReader 类的 EndOfStream 属性判断是否读取到文件末尾，如果没有，就调用 ReadLine()读取一行保存到 fileTexts 变量中。最后，输出内容并关闭文件流。

2. StreamWriter类

要在一个顺序文件中写入数据可以用 StreamWriter 类来实现。StreamWriter 类可以用一种特定的编码向字节流中写入字符。下面列出了 StreamWriter 类中常用的属性。

- AutoFlush：获取或设置一个值，指示 StreamWriter 是否在每次调用 StreamWriter.Write 之后，将其缓冲区刷新到基础流。
- BaseStream：获取同后备存储区连接的基础流。
- Encoding：获取将输出写入到其中的 Encoding。
- FormatProvider：获取控制格式设置的对象。
- NewLine：获取或设置由当前 TextWriter 使用的行结束符字符串。

下面的代码演示了如何使用 StreamWriter 类写入文件：

```
//指定要读取文件的路径
string fileName = Server.MapPath("/all.txt");
//判断文件是否存在
if (File.Exists(fileName))
{
    //创建写入流
    StreamWriter writer =
      new StreamWriter(fileName, true, System.Text.Encoding.Default);
    //使用流向文件中写入一行字符串
    writer.WriteLine("不要和我比懒，我懒得和你比。");
    //更新文件内容
    writer.Flush();
    //关闭写入流
    writer.Close();
}
```

在上述代码中，初始化了一个 StreamWriter 类对象 swriter，然后调用 StreamWriter 类中的 WriteLine()方法向文件中写入一行字符串，接下来的 Flush()方法更新了文件的内容，最后调用 StreamWriter 类的 Close()方法关闭写入流。

Web程序员成功之路

11.12.2　实例描述

今天遇到个小问题，上面给了一个任务说要进行员工岗位考核，想让出些与工作项目相关的题目，或者具有技术性、艺术性和难度的实战练习。凡是能与艺术沾点边的，都不简单。

为此，想到最近几个项目中对文件操作的使用频率越来越多，可以从这方面出一个。为了体现出“艺术性”的要求，在题目名称上不能太随意。经过讨论，决定使用时下最流行的游戏来作为话题，再加上文件技术，那就做一个在线故事接龙的游戏吧。

作为出题与技术考核员，首先得把标准答案给制订出来。下面是基本实现操作，答案则可以在此基础上自由扩展。

11.12.3　实例应用

【例 11-12】实现在线故事接龙游戏。

实现在线故事接龙游戏的具体制作步骤如下。

(1)　新建一个 ASPX 页面，添加一个 Repeater 控件来显示故事列表。这里通过表格简单修饰了一下，生成的布局代码如下：

```
<table width="100%">
    <asp:Repeater ID="Repeater1" runat="server">
    <ItemTemplate>
    <tr>
        <td>
        <img src='images/072.gif' width='9' height='8' />
        <%# Eval("Content") %>
        </td>
    </tr>
    <tr>
        <td height='1' background='images/073.gif'></td>
    </tr>
    </ItemTemplate>
    </asp:Repeater>
</table>
```

(2)　添加一个 TextBox 控件，将 TextMode 属性设置为 MultiLine，使它可以输入多行文本。接下方添加一个 Button 控件，用于保存：

```
<fieldset>
    <legend>编写一个新故事</legend>
    <asp:TextBox ID="TextBox1" runat="server" Height="100px"
      TextMode="MultiLine" Width="580px"></asp:TextBox>
    <br />
    <asp:Button ID="Button1" runat="server" Text="保存"
      OnClick="Button1_Click" />
</fieldset>
```

如图 11-25 所示是完成后的页面布局效果。

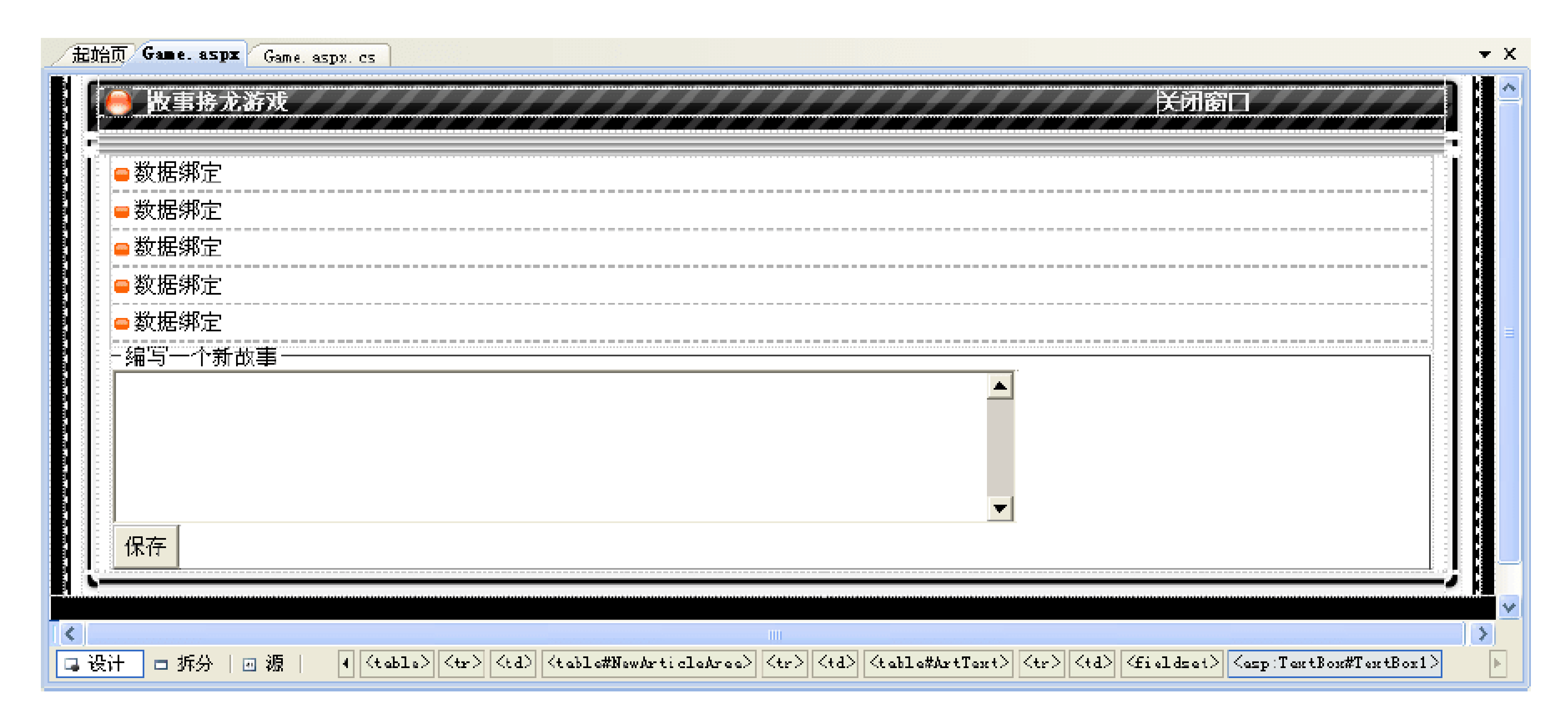


图 11-25　制作前台布局

(3) 编写第 1 段代码，使页面加载完成后显示故事列表：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        this.GetAllStory();       //加载故事列表
    }
}
```

(4) 打开“保存”按钮的单击事件，在这里需要判断内容是否为空。如果不为空，则追加到文件中，并显示最新内容：

```
protected void Button1_Click(object sender, EventArgs e)
{
    //如果输入的内容为空，直接返回
    if (this.TextBox1.Text.Trim() == string.Empty) return;
    FileInfo DBFile = new FileInfo(Server.MapPath("~/Story.txt"));
    StreamWriter sw = DBFile.AppendText();      //以追加的方式打开该文件
    sw.WriteLine(this.TextBox1.Text);        //写入内容
    sw.Close();                              //关闭流
    this.GetAllStory();                      //显示最新内容
}
```

(5) 添加前面所需的辅助方法 GetAllStory()，它的作用就是把保存在文件中的所有故事绑定到前台 Repeater 控件中进行显示：

```
/// <summary>
/// 绑定故事接龙列表
/// </summary>
private void GetAllStory()
{
    FileInfo DBFile = new FileInfo(Server.MapPath("~/Story.txt"));
    StreamReader sr = DBFile.OpenText();
```

```
    IList<Story> solitaireList = new List<Story>();
    string content = string.Empty;

    //遍历文件，封装实体
    while ((content = sr.ReadLine()) != null)
    {
        solitaireList.Add(new Story() { Content = content });
    }
    sr.Close();
    this.Repeater1.DataSource = solitaireList;
    this.Repeater1.DataBind();
}
```

(6) 声明一个用来封装故事信息的实体类，名称为 Story，类的完整定义如下：

```
class Story
{
    /* 故事实体 */
    string content;                //故事内容
    public string Content
    {
        get { return content; }
        set { content = value; }
    }
}
```

11.12.4　运行结果

运行 ASPX 页面后，如果是第 1 次运行，Story.txt 的内容为空，则看不到列表。可以在多行文本框里输入故事内容，之后单击“保存”按钮，就可以看到数据在上面列了出来。如图 11-26 所示。

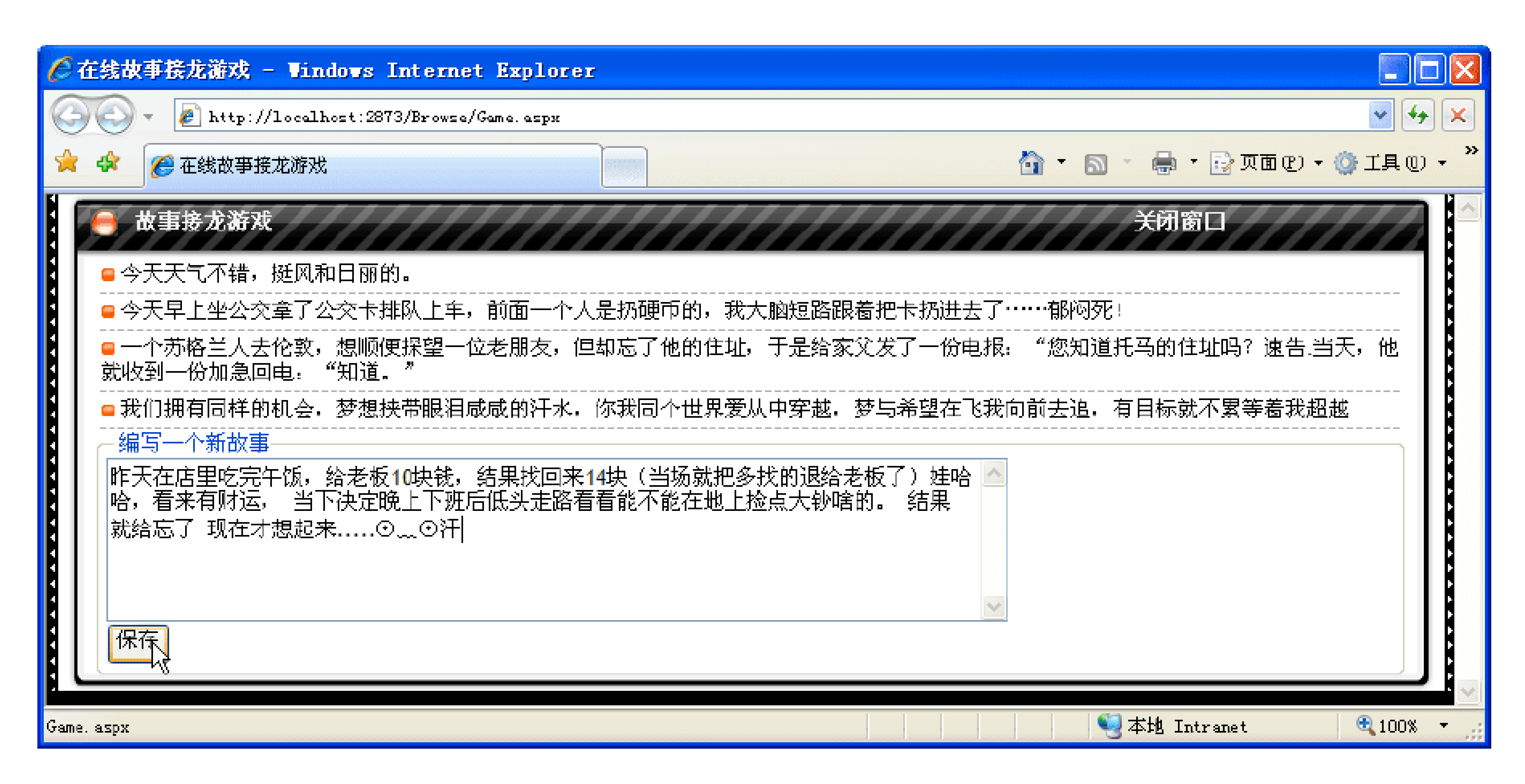


图 11-26　在线故事接龙的运行效果

11.12.5 实例分析



源码解析：

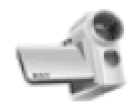
本实例主要运用了文件操作(FileInfo)和读写流(StreamReader, StreamWriter)技术来操作和读写文件。

在这里着重强调的是，每次创建“流”对象都必须关闭。如 StreamReader 和 StreamWriter 都要使用 Close()方法进行关闭。否则下次使用该资源时将会报出一个资源被占用异常。

运行本实例时需要在站点根目录下建立一个名为 solitaire.txt 的文本文件。

11.13 获取注册表的启动项

在应用程序安装时，常常需要利用注册表来记录应用程序的名称、运行路径、用户对应用程序的配置信息等。许多应用程序在运行时也常常需要访问注册表，获取系统或者程序启动相关信息等。



视频教学： 光盘/videos/11/RegistryKey.avi　　长度：7 分钟

11.13.1 基础知识——注册表操作类

在 Microsoft.Win32 命名空间中提供了两个类来操作注册表：Registry 类和 RegistryKey 类，并且这两个类都是密封类，不允许被继承。

1. Registry类

Registry 类提供表示 Windows 注册表中根项 RegistryKey 的对象，并提供访问项/值对的静态方法。Registry 类之所以能够访问注册表的项/值，这是因为注册表是一个存储设备，包含有关应用程序、用户和默认系统设置的信息。

Registry 类提供了注册表 7 个基项的公开对象。

- ClassesRoot：存储有关类型及其属性的信息，读取基项 HKEY_CLASSES_ROOT。
- CurrentUser：存储有关用户首选项的信息，读取基项 HKEY_CURRENT_USER。
- LocalMachine：存储本地计算机的配置信息，读取基项 HKEY_LOCAL_MACHINE。
- DynData：包含动态注册表数据，读取基项 HKEY_DYN_DATA。
- PerformanceData：存储软件组件性能信息，读取基项 HKEY_PERFORMANCE_DATA。
- Users：存储有关默认用户配置的信息，读取基项 HKEY_USERS。
- CurrentConfig：存储非用户特定的硬件信息，读取基项 HKEY_CURRENT_CONFIG。

一旦使用 Registry 类标识了要操作注册表的基项后，便可以使用 RegistryKey 类添加或移除子项和操作给定项的值。此外，Registry 类也提供了如下两个静态方法。

(1)　GetValue()方法

GetValue()方法的定义如下：

```
public object GetValue(string name);
public object GetValue(string name, object defaultValue);
```

参数 name 表示键的名称，返回类型是一个 object 类型，如果指定的键不存在则返回 null。如果失败又不希望返回的值是 null 则可以指定参数 defaultValue，指定了参数则在读取失败的情况下返回该参数指定的值。

(2)　SetValue()方法

SetValue()方法的定义如下：

```
public static void SetValue(string keyName, string valueName, Object value);
public static void SetValue(
string keyName, string valueName, Object value, RegistryValueKind valueKind);
```

SetValue 方法用于设置指定的注册表项的指定值。如果指定的项不存在，则创建该项。keyName 参数是以有效注册表根(如 HKEY_CURRENT_USER)开头的完整注册表路径。valueName 参数是值的名称。value 参数为要存储的数据。valueKind 参数是存储数据时使用的注册表类型。

技术文档	注册表简介

注册表的层次结构非常类似于文件系统。查看和修改注册表内容的一般方式是使用 regedit 或 regedt32 工具。其中，regedit 在所有的 Windows 版本中都有，而 regedt32 则在 Windows NT 和 Windows 2000 后的系统中才有。如图 11-27 所示为 XP 下运行 regedit 后打开的注册表编辑器。

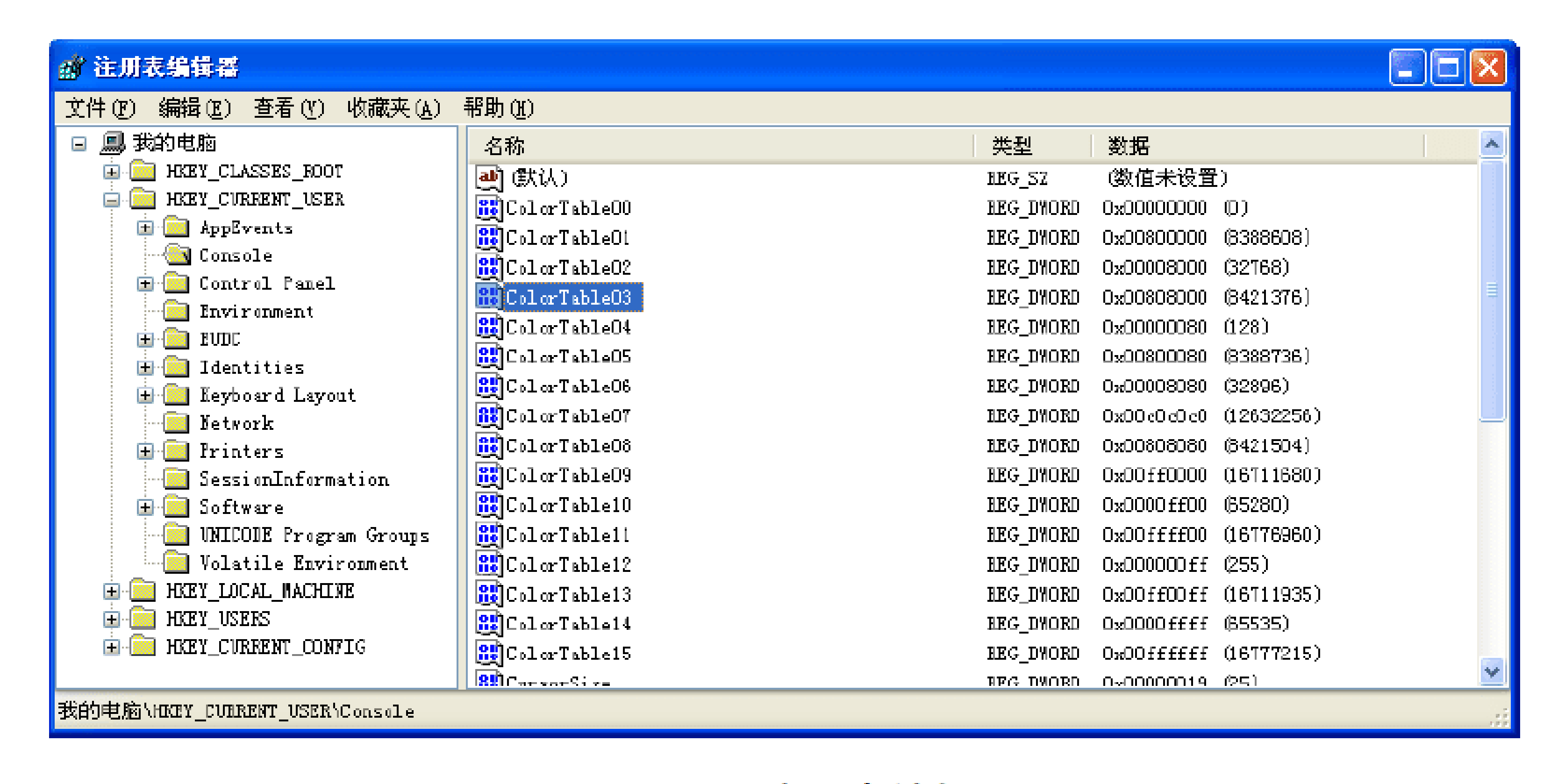


图 11-27　注册表编辑器

注册表编辑器非常类似于 Windows 资源管理器，与注册表本身的层次结构相匹配。在注册表中，最上面的节点是注册的根键，它们的位置是不能改变的。系统共有 7 个根键，但使用 regedit 只能看到其中的 5 个。

①　HKEY_CLASSES_ROOT(HKCR)

包含系统上文件类型的细节(.txt、.doc 等)，以及应用程序可以打开的文件类型。它还包

含所有 COM 组件的注册信息。

② HKEY_CURRENT_USER(HKCU)

包含用户目前登录的机器的设置。这些设置包括桌面设置、环境变量、网络和打印机连接及其他定义用户操作环境的设置。

③ HKEY_LOCAL_MACHINE(HKLM)

包含所有安装到机器上的软件和硬件信息，这些设置不是用户特有的，而是可为所有登录到机器上的用户所共享。它还包含 HKCR 根键：HKCR 实际上并不是一个独立的根键，而只是一个对注册表键 HKLM/SOFTWARE/Classes 的映射。

④ HKEY_USERS(HKUSR)

包含所有用户的用户配置。它还包含 HKCU 根项，HKCU 根项是对 HKEY_USERS 中一个键的映射。

⑤ HKEY_CURRENT_CONFIG(HKCF)

包含机器上的硬件配置信息。

每个注册表键都类似于文件系统中的文件夹或文件。但是，文件系统可以区分文件(文件中包含数据)和文件夹(其中主要包含其他文件夹或文件)，而注册表中只有键。键可以包含数据和其他的键。

如果键中包含数据，这个键就表示为一组值。每个值都有一个相关的名称、数据类型和数据，另外，键还可有默认值。键的数据类型分别为 REG_SZ(字符串类型)、REG_DWORD(无符号的 32 位整数)和 REG_BINARY(字节类型)。

2. RegistryKey类

RegistryKey 封装了对注册表基本操作的方法，包括读取、写入和删除等。也提供了很多的属性来获得注册表的信息，表 11-10 列出了这些方法和属性。

表 11-10　RegistryKey类的成员

成员名称	说　明
Close 方法	关闭该项，如果该项的内容已修改，则将该项刷新到磁盘
CreateSubKey 方法	创建一个新子项或打开一个现有子项
DeleteSubKey 方法	删除指定的子项。字符串 subkey 不区分大小写
DeleteSubKeyTree 方法	递归删除子项和任何子级子项。字符串 subkey 不区分大小写
DeleteValue 方法	从此项中删除指定值
Flush 方法	将指定的打开注册表项的全部属性写到注册表中
GetAccessControl 方法	返回当前注册表项的访问控制安全性
GetSubKeyNames 方法	检索包含所有子项名称的字符串数组
GetValue 方法	检索与指定名称关联的值
GetValueKind 方法	检索与指定名称关联的值的注册表数据类型
GetValueNames 方法	检索包含与此项关联的所有值名称的字符串数组
OpenRemoteBaseKey 方法	打开一个新的 RegistryKey，表示远程计算机上的请求的项

续表

成员名称	说　明
OpenSubKey 方法	检索指定的子项
SetAccessControl 方法	向现有注册表项应用 Windows 访问控制安全性
SetValue 方法	设置注册表项中的名称/值对的值。 从所存储数据的类型或指定的 RegistryValueKind 确定注册表数据类型
Name 属性	检索项的名称
SubKeyCount 属性	检索当前项的子项数目
ValueCount 属性	检索项中值的计数

使用该类在对注册表进行操作时，必须具有系统提供的权限。否则将不能完成指定的操作，程序也将抛出异常。

(1)　创建子键

创建子键的方法声明格式如下：

```
public RegistryKey CreateSubKey(string subkey)
```

其中，参数 subkey 表示要创建的子键名称或者子键全路径，此字符串不区分大小写。如果创建成功，返回值就是被创建的子键，一个 RegistryKey 对象；否则操作失败则为 null。

例如，在注册表的 Users 根下创建一个名为 TestRegistryKey 的子键，使用代码如下：

```
RegistryKey rootKey = Registry.Users;
rootKey.CreateSubKey("TestRegistryKey");
```

(2)　打开子键

OpenSubKey()方法可以打开指定子键，语法声明如下：

```
public RegistryKey OpenSubKey(string name);
public RegistryKey OpenSubKey(string name, bool writable);
```

参数 name 表示要打开子键的名称或者子键全路径，writable 参数表示被打开的主键是否可以被修改。其中，第 1 个方法默认对打开的子键是只读的，如果要对打开的主键进行写操作，可使用第 2 个方法并设置 writable 参数为 true。

该类中还为我们提供了另一个方法，用于打开远程计算机上的注册表。如下所示是该方法的语法：

```
public static RegistryKey OpenRemoteBaseKey(RegistryHive hKey,
  string machineName);
```

下面的代码介绍了如何使用它们来打开一个键：

```
RegistryKey rootKey = Registry.Users;
RegistryKey newKey = rootKey.OpenSubKey("TestRegistryKey");
//打开远程计算机 remoteName 上的 HKEY_CURRENT_USER\Environment 键
RegistryKey environmentKey = RegistryKey.OpenRemoteBaseKey(
  RegistryHive.CurrentUser, remoteName).OpenSubKey("Environment");
```

(3) 删除子键

DeleteSubKey()方法用于删除指定的键，语法声明如下：

```
public void DeleteSubKey(string subkey);
```

使用 DeleteSubKey()方法时，如果键中还包含子键，则会删除失败，并返回一个异常。如果要彻底删除键的目录，即删除键以及下面包含的所有子键，可以使用 DeleteSubKeyTree()方法，它的语法如下：

```
public void DeleteSubKeyTree(string subkey);
```

下面的代码演示了如何删除键：

```
RegistryKey rootKey = Registry.Users;
rootKey.DeleteSubKey("TestRegistryKey");
rootKey.DeleteSubKeyTree("TestRegistryKey");
```

(4) 读取键值

读取键值方法的语法格式如下：

```
public object GetValue(string name);
public object GetValue(string name, object defaultValue);
```

参数 name 表示键的名称，返回类型是一个 object 类型。如果方法中指定的键不存在，则方法返回一个 null。我们在使用 GetValue()方法时，可以不必关心该键值的值类型是字符串、二进制或者 DWORD 类型，只要使用正确的返回类型就可以了。例如，我们要读取一个字符串类型的键，代码就可以这样写：

```
string s_value = key.GetValue("Type");
```

这里 key 表示一个主键。如果不确定键值是否存在，又不希望返回 null 值。那就可以调用第 2 个 GetValue()方法，其中 defaultValue 参数表示默认的返回值，如果读取失败，返回值就是传递给 defaultValue 参数的值。

(5) 设置键值

设置键值用 SetValue()方法，下面是语法格式：

```
public void SetValue(string name, object value);
```

同样，我们在使用该方法修改键值时，也不用理会该传递哪种值类型。该方法将会自动识别键类型，将把相应类型的值赋予指定的值。

下面的代码详解了使用 SetValues()方法设置键值的过程：

```
//创建一个新的键，名称为 RegistrySetValueExample
RegistryKey rootKey =
  Registry.CurrentUser.CreateSubKey("RegistrySetValueExample");
//设置 long 类型值
rootKey.SetValue("LargeNumberValue1", (long)42);
//设置大数值
rootKey.SetValue("LargeNumberValue2", 42000000000);
//设置 DWord 类型值
```

```
rootKey.SetValue("DWordValue", 42);
//设置字符串数组
rootKey.SetValue("MultipleStringValue",
  new string[] {"One", "Two", "Three"});
//设置字节数组
rootKey.SetValue("BinaryValue", new byte[] {10, 43, 44, 45, 14, 255});
// 设置字符串
rootKey.SetValue("StringValue", "The path is %PATH%");
```

11.13.2　实例描述

在注册表中查看系统的启动项。相信懂点 Windows 系统操作的人都会说，“这可是超级简单啊。开机后 360 工具就会显示了，或者运行 regedit 打开看看就更直接了。都什么年代了，放着好好的工具不用，谁会记那些东西呢？”可是，读者注意了，万变不离其宗，只有掌握其本质，才能灵活应用到任何场景中。而且现在可是在 ASP.NET 页面中的实现啊。

11.13.3　实例应用

【例 11-13】获取注册表的启动项。既然用 ASP.NET 实现，那首先总得有一个 ASPX 页面吧。另外，注册表中的启动项信息可能有很多，如何显示呢？

在页面上添加一个 ListBox 控件，用于显示启动项信息，设置宽为 560，高为 160，如下是添加后生成的页面代码：

```
<asp:ListBox ID="ListBox1" runat="server" Width="560" Height="160" >
</asp:ListBox>
```

转到后台代码.cs 文件，然后找到 Page_Load()事件，编写在 ListBox 中显示可用驱动器列表的代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    //初始化 rootKey，操作 HKEY_LOCAL_MACHINE 主键
    RegistryKey rootKey = Registry.LocalMachine;
    //初始化 runKey，操作 Software\Microsoft\Windows\CurrentVersion\Run 键
    RegistryKey runKey =
    rootKey.CreateSubKey(@"Software\Microsoft\Windows\CurrentVersion\Run");
    //初始化属性值名字的列表
    string []names = runKey.GetValueNames();
    //遍历名字列表
    foreach (string valueName in names)
    {
        this.ListBox1.Items.Add("启动项名称：" + valueName);
        this.ListBox1.Items.Add("值：" + runKey.GetValue(valueName));
        this.ListBox1.Items.Add("");
    }
}
```

这样三两下就完成了，很简单吧。主要是利用了 RegistryKey 和 Registry 类，它们位于 Microsoft.Win32 命名空间，记得要引用。

11.13.4 运行结果

现在运行一下程序。非常顺利，可以在 ListBox 中看到系统注册表里启动项的名称和值，如图 11-28 所示。

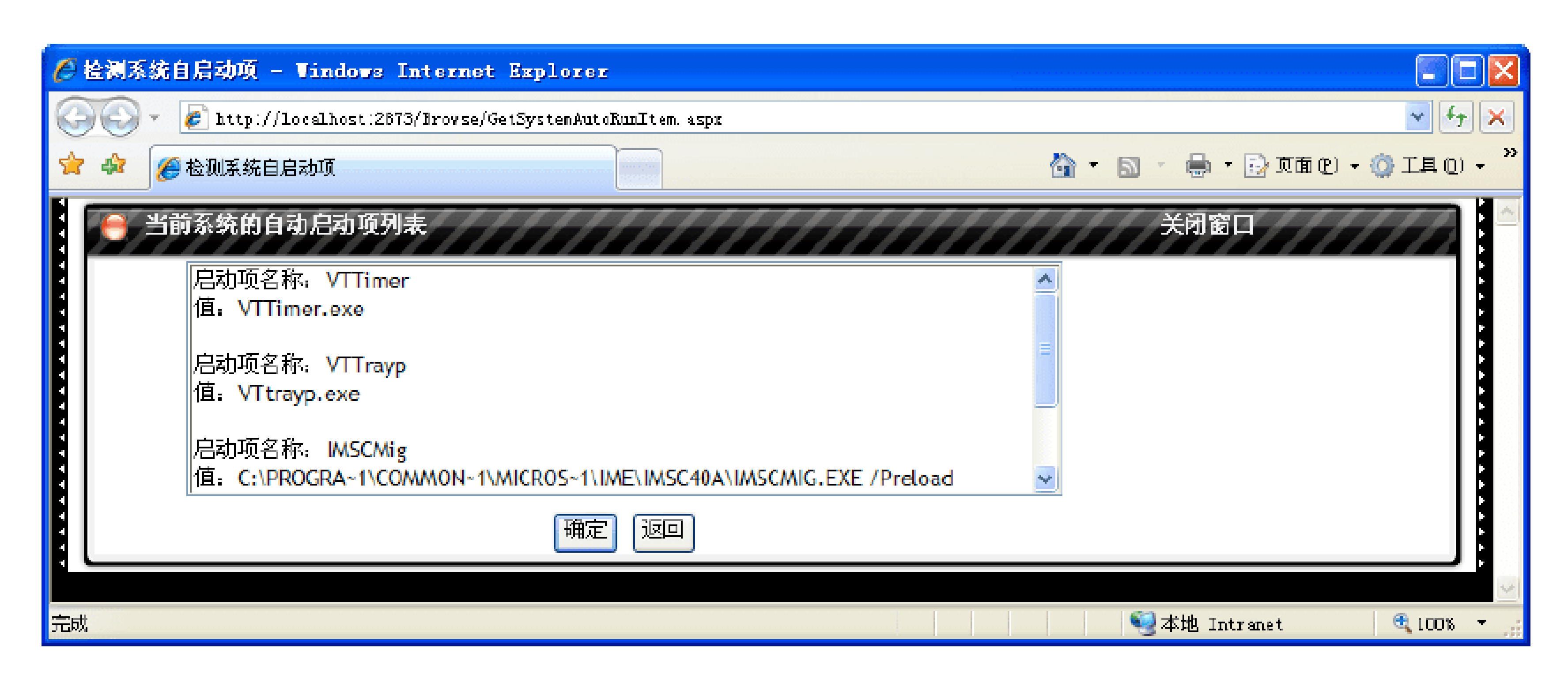


图 11-28　查看启动项和值

11.13.5 实例分析



源码解析：

实现本实例时，需要了解基本的注册表知识，否则会难以理解该程序。

NET 以 Registry 类的属性的方式提供了注册表的 7 个根键，由这 7 个根键创建并获取其下的子键，并对其操作。

要注意页面 cs 文件里需要引入所需命名空间：Microsoft.Win32。另外，RegistryKey 类还提供了 GetSubKeyNames()方法获取子键名称列表，可进一步地对注册表键进行操作。

最后提醒用户：操作注册表时请多加注意，以免执行误操作导致操作系统崩溃。

11.14 常见问题解答

11.14.1 StreamReader无法读取中文命名的Txt文件



StreamReader 无法读取中文命名的 Txt 文件？

网络课堂：http://bbs.itzcn.com/thread-2976-1-1.html

StreamReader 无法读取中文命名的 Txt 文件：

```
StreamReader myreader = new StreamReader("中文.txt",
  System.Text.Encoding.GetEncoding("gb2312"));
```

文件路径是对的，为什么不能读取？为什么文件重命名为英文却可以？

【解决办法】：这个问题作者也遇到过，改了一下 web.config 就可以了。

```
<globalization requestEncoding="gb2312" responseEncoding="gb2312" />
```

11.14.2　如何判断上传的图片，在服务器文件夹里已经有了此图片



如何判断上传的图片，在服务器文件夹里已经有了此图片？
网络课堂：http://bbs.itzcn.com/thread-2962-1-1.html

如何判断上传的图片？在服务器文件夹里已经有了此图片。

主要想用文件名来判断，或者其他办法也行，请各位帮帮忙！可以详细点吗，我的存储路径为 D:\ASP.NET\Lab_6\Images，请大家帮帮忙啊。

【解决办法】：很简单，直接调用 File 类的 Exists()方法就行了。它可以用来判断文件名，语法如下：

```
string strFileName = @"D:\ASP.NET\Lab_6\Images";
//判断文件是否存在
if (System.IO.File.Exists(strFileName))
{
    //在这里编写你的代码
}
```

呵呵，需要学的还很多呢，大家加油！

11.14.3　怎样在指定文件夹下显示所有图片路径和名称



怎样在指定文件夹下显示所有图片路径和名称？
网络课堂：http://bbs.itzcn.com/thread-2965-1-1.html

编写一个程序，选择某个文件夹，则将此文件夹下的所有图片显示在列表中(包括路径和名称)，并生成 TXT 文档(一张图片生成一个 TXT 文档)，命名为“图片名称.txt”。

【解决办法】：使用下面的几行代码就能搞定了：

```
System.IO.DirectoryInfo di = new System.IO.DirectoryInfo("文件夹路径");
FileInfo []ff = di.GetFiles("*.jpg");       //di 文件夹下的全部 jpg 文件
foreach (FileInfo temp in ff)
{
    File.Create(temp.FullName + ".txt");    //创建 txt 文档
}
```

11.14.4 ASP.NET如何读取文件夹下的所有图片名称



ASP.NET 如何读取文件夹下所有图片名称？
网络课堂：http://bbs.itzcn.com/thread-2959-1-1.html

ASP.NET 如何读取文件夹下所有图片的名称？这个具体的方法是怎么调用的。谢谢

【解决办法】：问这样问题的人看来不是很懂，其实并不难，不信试试下面的代码，注意改一下路径啊：

```
System.IO.DirectoryInfo di =
  new System.IO.DirectoryInfo(@"E:\myPhotos\fengjing");
System.IO.FileInfo []fimore = di.GetFiles();
foreach (System.IO.FileInfo fi in fimore)
{
    Response.Write(fi.Name);
    Response.Write("<br>");
}
```

11.14.5 ASP.NET文件上传的最大限制是多少



ASP.NET 文件上传的最大限制是多少？
网络课堂：http://bbs.itzcn.com/thread-2969-1-1.html

请教高手，ASP.NET 文件上传最大限制是多少？在哪里设置文件上传大小的限制？我现在要给一个网站开允许上传 40MB 可以做到吗？

【解决办法】：40MB 太夸张了吧。ASP.NET 默认只允许上传 2MB 文件，在 web.config 中增加如下配置，一般可以自定义最大文件大小：

```
<httpRuntime
  executionTimeout="300"
  maxRequestLength="40960"
  useFullyQualifiedRedirectUrl="false"/>
```

其中 maxRequestLength 属性就是限制上传大小的，如设为“40960”即为 40MB。

11.15 习　　题

一、填空题

(1) 判断一个文件是否存在可以使用 File 类中的__________方法。
(2) __________类提供了实现创建、移动和枚举文件的实例方法。
(3) 设置当前工作目录为“E:\C#\2010”的代码是:

```
string psd=Directory.___________;
if(psd!=_________)
Directory.___________;
```

(4) 使用 StreamWriter 类实现顺序文件的__________操作，__________类实现顺序文件的读取操作。

(5) 对注册表操作的__________类和 RegistryKey 类位于__________命名空间，并且这两个类都是__________类，不允许被继承。

二、选择题

(1) ASP.NET 对于文件的操作位于__________命名空间。

A. system.io　B. system.file　C. system.text.in　D. system.web.file

(2) 要实现删除一个文件的功能，下面哪些类可以实现？__________

A. Directory　B. DirectoryInfo　C. File　D. FileInfo

(3) __________类是用来公开创建、移动和枚举目录和子目录的静态方法。

A. Directory　B. DirectoryInfo　C. File　D. FileInfo

(4) 使用 Directory 类的 Delete()方法时引发了 ArgumentNullException 异常，引起此异常的原因为__________。

A. 参数超出了系统定义的最大长度　B. 参数是一个文件名

C. 参数无效　D. 参数为空引用

(5) 下面不属于 Registry 类提供方法的是__________。

A. CreateSubKey()方法　B. Flush()方法

C. GetValue()方法　D. OpenRemoteBaseKey()方法

(6) CreateDirectory 方法接受一个参数，表示要创建的目录，该方法返回一个 DirectoryInfo 类实例，表示新创建的目录或子目录。如下代码为正确使用该方法的选项代码的是____。

A.
```
DirectoryInfo di;
di = Directory.CreateDirectory("E:\\C#");
```
B.
```
DirectoryInfo di;
di = Directory.CreateDirectory("C#");
```
C.
```
FileInfo fi;
fi = Directory.CreateDirectory("E:\C#");
```
D.
```
FileInfo fi
fi = Directory.CreateDirectory("E:\\C#");
```

三、上机练习

上机练习：远程管理文件。

在电子商务网站中，远程管理文件是一个常用的功能，可以远程删除文件、重命名文件、查看文件相关信息和上传文件。本次上机练习要求：应用 System.IO 命名空间和 FileUpload 控件完成文件管理页面。

最终运行后，在页面可以上传文件、查看文件信息、删除文件和重命名文件等，效果如

图 11-29 所示。

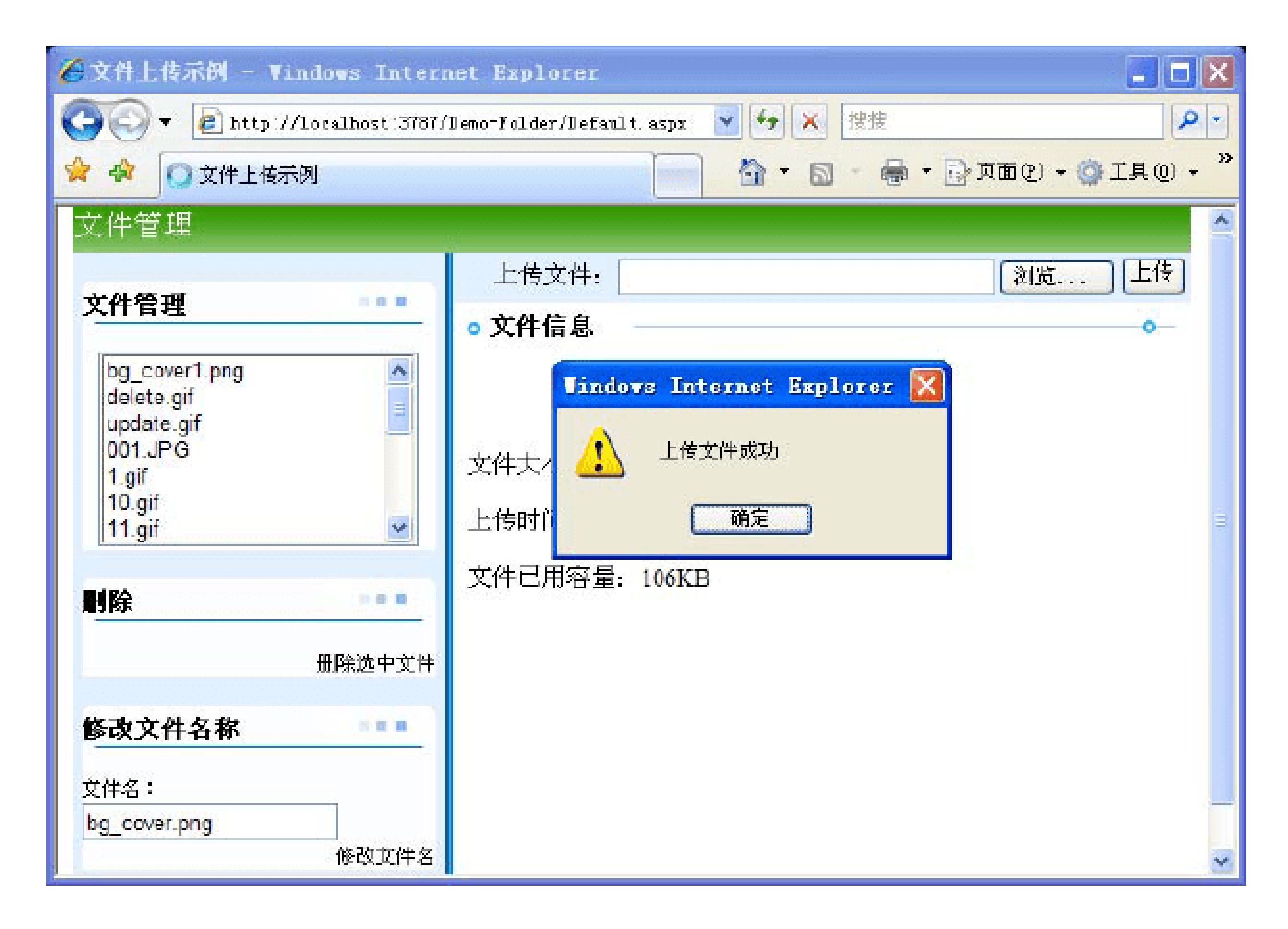


图 11-29　远程管理文件的效果



第 12 章　缝缝补补的ASP.NET

内容摘要：

自 Web 诞生以来，人们就在尽力让它运行得更快。发展到今天，Web 获得了巨大的成功。但是这也意味着有更多的用户加入进来、有更多的数据加入进来。当然由于数据量越来越庞大，人们从 Web 上获取数据也需要花费越来越多的时间。所以，如何将 Web 程序的性能调至最佳状态，是 Web 应用程序开发人员不得不考虑的问题。

Windows Server 和 IIS 本身具有完善的安全机制。可是，Microsoft 利用许多复杂的系统底层功能设计了.NET Framework 和 ASP.NET，这些复杂的系统底层功能也很大程度地涉及到了系统安全性，并且这些设计的安全性也是保障我们应用程序系统安全的关键组成部分。这就意味着，ASP.NET 应用程序开发人员和系统管理员都要负责保护 Web 应用程序的安全。

所以，本章就从运行效率和安全性两方面来优化一下我们的 Web 应用程序。

学习目标：

- 了解 ASP.NET 的底层运行机制
- 掌握常用的数据安全技术
- 掌握程序代码的常用优化技巧
- 掌握 ASP.NET 程序的性能优化方法
- 掌握对数据访问操作的优化方法
- 掌握单元测试工具 NUnit 的使用方法
- 了解 ASP.NET 的身份验证方式

12.1　数据安全技术

在 Web 站点中，安全性管理常常是一个相当复杂的事情。因为我们需要考虑许许多多的因素，比如：

- 在保证简便的使用性的前提下阻止非法用户的访问。
- 确保数据库数据的安全性。
- 阻止资源文件的非法使用。
- 最大程度地保护用户的隐私。

在 ASP.NET 中，有一套完整的方案来解决这个问题，下面是一些做法。

(1) 使用登录用户来区分不同权限的访客。

(2) 对操作数据库的命令进行验证和过滤来保证数据库的安全。

(3) 使用 ASP.NET 提供的 HttpHandler 来对资源文件的请求进行验证。

(4) 使用数据加密的方式，在即使系统崩溃的情况下仍然能保证一些重要数据不被泄露。

12.1.1　自定义加密

数据加密技术早在二战时期都已经发挥得淋漓尽致。

我们看一些二战时期的影视剧时，经常会见到有人通过电报或广播传递一些情报，但是传递的报文让观众看了总摸不着头脑。

比如有一次看到八路的地下党接到一个电报，报文是“雨天屋漏，赶紧带着孩子回娘家看看”，接着那伙计就赶紧收拾东西跑了，不一会儿他的屋门就被敌人踹开了……后来才知道那是他们同事发的密报，意思是“有人把你出卖了，你已经暴露，赶紧撤回根据地吧”。

这样的电报就是让敌人看到，也不会有什么破绽，效果确实不错。

这就是数据加密技术的一个小例子。

数据加密，简单地说，就是根据一套规则，对信息进行加工、混淆，让不知道这个规则的人无法知道信息的原意。

下面我们就亲自动手，来了解一下数据加密。

 **视频教学：光盘/videos/12/CustomEncode.avi**　　**长度：6 分钟**

1. 实例描述

自定义数据加密，首先要先自己定义一套加密的规则。

这个实例中，我们就先对输入的内容进行关键字替换，然后对整个信息内容进行反转，然后把字符串中的每一个英文字母的 ASCII 码加 1。得到的字符串就是我们加密过的信息。

当然，在解密的时候我们要先把字符串中的每个英文字母的 ASCII 码值减去 1，然后进行倒序，然后进行关键字反替换，就得到我们原来的明文信息了。

2. 实例应用

【例 12-1】自定义加密。

(1) 新建一个 Web 项目。

(2) 打开 Default.aspx 页面，在视图下拖入一个文本框，用于输入待加密的数据，拖入一个命令按钮，用于触发编码操作，再放入一个标签控件，用于展示编码过的内容。代码如下：

```
<asp:TextBox ID="txtContent" runat="server" TextMode="MultiLine"
  Height="60px" Width="350px"></asp:TextBox>
<br />
<asp:Button ID="btnEncode" runat="server" Text=" 编码 "
  onclick="btnEncode_Click" />
<br />
<asp:Label ID="lblResult" runat="server" Width="350" ></asp:Label>
```

(3) 打开页面后台代码文件，添加 3 个方法，分别执行加密和解密的替换关键字、反转顺序、ASCII 编码操作。代码如下：

```
private string ReplaceKeyword(string code, bool isEncode)
{
    /* 替换关键字 */

    string []key1 = { "hello", "me", "apple" };
    string []key2 = { "nihao", "wo", "pingguo" };
    string []key, values;

    if (isEncode)
    {
        /* 如果是编码，key1 中的元素作键，key2 中的元素作值 */
        key = key1;
        values = key2;
    }
    else
    {
        /* 如果是解码，key2 中的元素作键，key1 中的元素作值 */
        key = key2;
        values = key1;
    }

    /* 替换关键字 */
    for (int i=0; i<key.Length; i++)
    {
        code = code.Replace(key[i], values[i]);
    }
    return code;
}

private string InvertedOrder(string code)
{
```

```
    /* 倒序字符串 */

    char[] cs = code.ToCharArray();    //得到一个字符数组
    Array.Reverse(cs);    //倒序排序该数组
    code = new string(cs);    //根据倒序过的字符数组，创建字符串
    return code;
}

private string ASCII(string code, bool isEncode)
{
    /* ASCII */

    int incremental = isEncode ? 1 : -1;    //确定当前操作是加 1 还是减 1
    string tempStr = string.Empty;    //临时字符串

    foreach (char c in code)
    {
        if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        {
            /* 如果是字母，执行操作 */
            tempStr += (char)(c + incremental);
        }
        else
        {
            /* 如果不是字母，直接使用 */
            tempStr += c;
        }
    }
    return tempStr;
}
```

(4) 给页面命令按钮添加单击事件，修改事件代码，执行相关的加密和解密的操作。代码如下：

```
protected void btnEncode_Click(object sender, EventArgs e)
{
    //hello, give me an apple

    this.lblResult.Text ="";    //清空标签
    string content = this.txtContent.Text;    //获取输入的内容

    /* 以下进行编码操作 */
    this.lblResult.Text += "<br><b>编码</b>";
    content = this.ReplaceKeyword(content, true);
    this.lblResult.Text += "<br>替换关键字后的结果：" + content;
    content = this.InvertedOrder(content);
    this.lblResult.Text += "<br>倒序以后的结果：" + content;
    content = this.ASCII(content, true);
    this.lblResult.Text += "<br>所有字母 ASCII 码加 1 后的结果：" + content;
```

```
    /* 以下进行解码操作 */
    this.lblResult.Text += "<br><b>解码</b>";
    content = this.ASCII(content, false);
    this.lblResult.Text += "<br>反向替换关键字后的结果：" + content;
    content = this.InvertedOrder(content);
    this.lblResult.Text += "<br>倒序以后的结果：" + content;
    content = this.ReplaceKeyword(content, false);
    this.lblResult.Text += "<br>所有字母 ASCII 码减 1 后的结果：" + content;
}
```

3. 运行结果

运行项目，访问 Default.aspx 页面。

在文本框中输入“hello,give me an apple”，然后单击“编码”按钮。执行结果如图 12-1 所示。

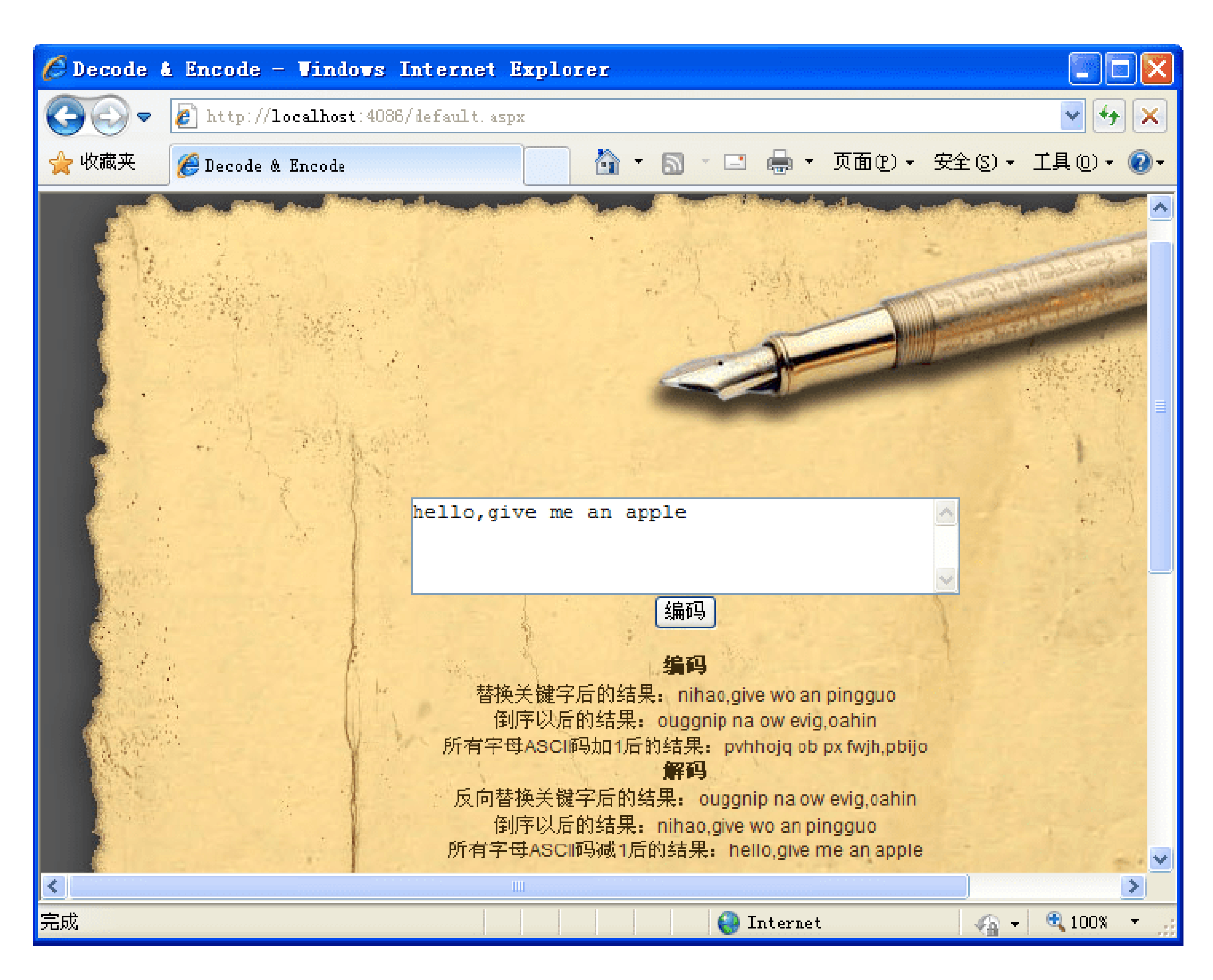


图 12-1　自定义加密执行结果

4. 实例分析



源码解析：

实例首先使用定义的替换词组对页面输入的内容进行替换，然后将字符串结果转换成字符数组，用得到的反转后的字符数组创建一个字符串，然后遍历得到的字符串结果，将其中的英文字母用每个字母的 ASCII 值加 1 进行替换。

由此我们得到一个乱七八糟的字符，任那破解之人再牛，也不可能用大脑猜出来我们加密前的原文是什么。

当然由于我们知道加密规则，所以能够根据加密规则反向地一步步解密。

12.1.2 对称加密

前面我们简单地演示了如何以自定义的算法进行数据加密。

ASP.NET 在 System.Security.Cryptography 命名空间下面提供了一系列现成的数据加密类库，来实现当前最流行的加密方法。

常见的加密方法有对称加密、不对称加密、加密签名、加密哈希等。其中使用频率最高的一个就是对称加密。



视频教学：光盘/videos/12/DES.avi 长度：6 分钟

1. 基础知识——对称加密DES

对称加密也叫密钥加密，指加密和解密使用相同密钥的加密算法。

它要求数据所有者和数据使用者在传递数据之前商定一个密钥。对称算法的安全性依赖于密钥，泄漏密钥就意味着任何人都可以对他们传递的数据解密，所以密钥的保密性对通信至关重要。

由于对称加密速度快，通常在需要加密大量数据时使用。对称加密算法是当今使用频率最高的加密算法。

对称加密方式的工作原理如图 12-2 所示。

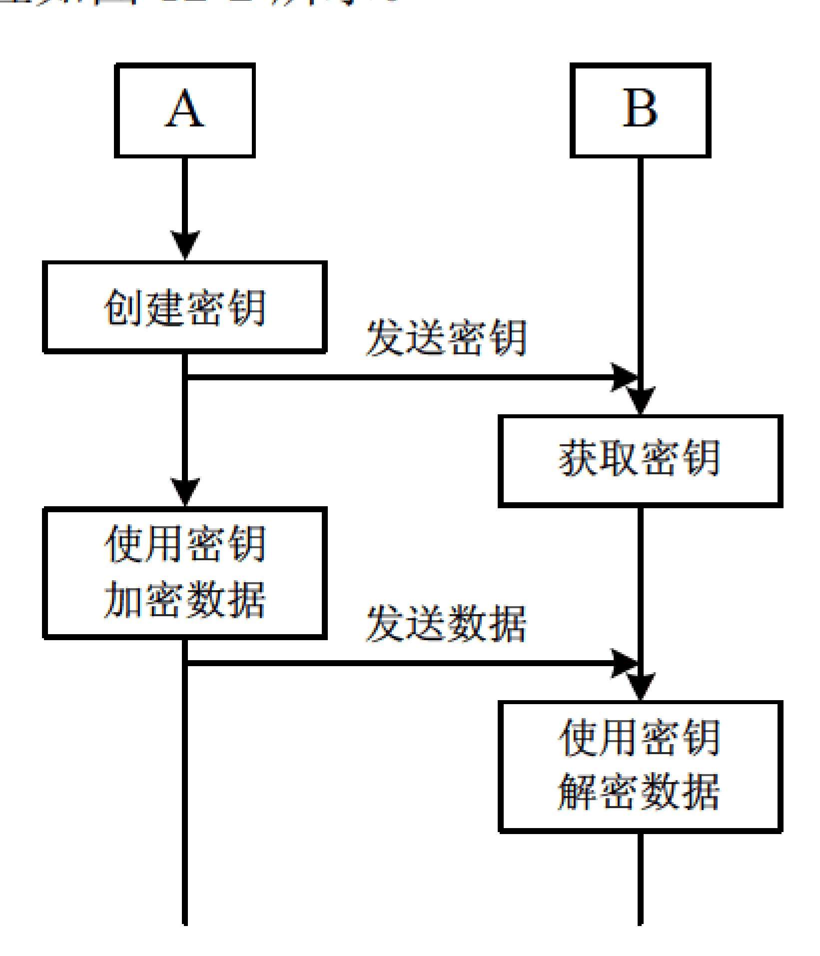


图 12-2 对称加密工作原理图



所谓对称，就是采用这种加密方法的双方使用方式用同样的密钥进行加密和解密。密钥是一种算法，数据所有者使用这种算法加密数据，数据使用者以同样的算法解密数据。因此对称加密本身是不安全的。

对称加密的常用加密算法有 DES、IDEA、RC2、RC4、RC5、SKIPJACK 和 Blowfish 等。以下我们以 DES 为例，来了解一下对称加密。

DES(Data Encryption Standard)是当前最著名的对称密钥加密算法。该算法是由 IBM 公司在 20 世纪 70 年代发展起来的，经过美国政府的加密标准筛选后，于 1976 年 11 月被政府采用，

随后又被美国国家标准局和美国国家标准协会(American National Standard Institute，ANSI)承认。

在 ASP.NET 的 System.Security.Cryptography 命名空间下有一个 DESCryptoServiceProvider 类，该类可以创建一个使用对称加密标准 DES 算法进行加密解密的对象。

该类有两个属性，用来设置加密的密钥和初始化向量。

- Key：指定加密的密钥。
- IV：指定初始化向量。



该算法支持长度为 64 位的密钥。每个半角的字符是 8 位，所以这里可以设置 8 个半角的字符作为密钥。

该类有两个方法，分别用来创建加密器对象和解密器对象。

- CreateDecryptor：创建对称数据加密标准(DES)解密器对象。
- CreateEncryptor：创建对称数据加密标准(DES)加密器对象。

因为.NET 的 DES 算法加密只能对数据流中的数据进行加密，所以这里还要用到内存流(MemoryStream)对象和加密流(CryptoStream)对象。

前面我们在学习 IO 的时候，了解过“流”的概念，在这里基本都一样，就不详细介绍了。具体使用方法来看实例。

2. 实例描述

本实例我们编写一个用于对 DES 算法进行加密和解密的工具。

该工具可以将我们输入的原文内容加密成密文，也可以将加密过的密文解密成加密前的原文内容。

3. 实例应用

【例 12-2】对称加密。

(1) 打开前面我们使用的项目。

(2) 这里我们为使程序更加结构化，新建一个用 DES 算法进行加密解密的工具类，取名为 DESSecurity，该类有两个方法，一个用于根据传入的文本和密钥进行加密，一个用于根据传入的文本和密钥进行解密。具体代码如下：

```
public class DESSecurity
{
    public string DESEncrypt(string code, string sKey)
    {
        /* 创建一个 DES 加密服务提供者 */
        DESCryptoServiceProvider des = new DESCryptoServiceProvider();
        /* 将要加密的内容转换成一个 Byte 数组 */
        byte []inputByteArray = Encoding.Default.GetBytes(code);
        /* 设置密钥和初始化向量 */
        des.Key = ASCIIEncoding.ASCII.GetBytes(sKey);
        des.IV = ASCIIEncoding.ASCII.GetBytes(sKey);
        /* 创建一个内存流对象 */
        MemoryStream ms = new MemoryStream();
```

```
        /* 创建一个加密流对象 */
        CryptoStream cs =
          new CryptoStream(ms, des.CreateEncryptor(), CryptoStreamMode.Write);
        /* 将要加密的文本写到加密流中 */
        cs.Write(inputByteArray, 0, inputByteArray.Length);
        /* 更新缓冲 */
        cs.FlushFinalBlock();
        /* 获取加密过的文本 */
        StringBuilder ret = new StringBuilder();
        foreach (byte b in ms.ToArray())
        {
            ret.AppendFormat("{0:X2}", b);
        }
        /* 释放资源 */
        cs.Close();
        ms.Close();
        /* 返回结果 */
        return ret.ToString();
    }

    public string DESDecrypt(string code, string sKey)
    {
        /* 创建一个 DES 加密服务提供者 */
        DESCryptoServiceProvider des = new DESCryptoServiceProvider();
        /* 将要解密的内容转换成一个 Byte 数组 */
        byte[] inputByteArray = new byte[code.Length / 2];
        for (int x=0; x<code.Length/2; x++)
        {
            int i = (Convert.ToInt32(code.Substring(x * 2, 2), 16));
            inputByteArray[x] = (byte)i;
        }
        /* 设置密钥和初始化向量 */
        des.Key = ASCIIEncoding.ASCII.GetBytes(sKey);
        des.IV = ASCIIEncoding.ASCII.GetBytes(sKey);
        /* 创建一个内存流对象 */
        MemoryStream ms = new MemoryStream();
        /* 创建一个加密流对象 */
        CryptoStream cs =
          new CryptoStream(ms, des.CreateDecryptor(), CryptoStreamMode.Write);
        /* 将要解密的文本写到加密流中 */
        cs.Write(inputByteArray, 0, inputByteArray.Length);
        /* 更新缓冲 */
        cs.FlushFinalBlock();
        /* 返回结果 */
        return System.Text.Encoding.Default.GetString(ms.ToArray());
    }
}
```

(3) 我们创建一个执行加密解密操作的 Web 窗体，取名为 DES.aspx。

(4) 在页面中添加 5 个文本框控件，分别用于输入原文内容、加密密钥、加密结果、解密密钥、解密结果，再加入两个命令按钮控件，分别用于执行加密和解密操作。具体代码如下：

```
<table width="390" cellpadding="5" cellspacing="5" border="1">
<tr>
<td width="80">原文内容：</td>
<td>
<asp:TextBox ID="txtSource" runat="server" Height="80" Width="300"
  TextMode="MultiLine"></asp:TextBox><br />
<asp:TextBox ID="txtEncodeKey" runat="server"></asp:TextBox>(密钥)
</td>
</tr>
<tr>
<td>加密结果：</td>
<td>
<asp:TextBox ID="txtEncode" runat="server" Height="80" Width="300"
  TextMode="MultiLine"></asp:TextBox><br />
<asp:TextBox ID="txtDecodeKey" runat="server"></asp:TextBox>(密钥)
</td>
</tr>
<tr>
<td>解密结果：</td>
<td>
<asp:TextBox ID="txtDecode" runat="server" Height="80" Width="300"
  TextMode="MultiLine"></asp:TextBox>
</td>
</tr>
</table>

<asp:Button ID="btnEncode" runat="server" Text="加密"
  onclick="btnEncode_Click" /> 
<asp:Button ID="btnDecode" runat="server" Text="解密"
  onclick="btnDecode_Click" />
```

(5) 在页面后台代码中添加两个命令按钮的事件处理方法，分别执行加密和解密操作。代码如下：

```
protected void btnEncode_Click(object sender, EventArgs e)
{
    /* 执行加密操作 */
    DESSecurity des = new DESSecurity();
    this.txtEncode.Text =
      des.DESEncrypt(this.txtSource.Text, this.txtEncodeKey.Text);
}

protected void btnDecode_Click(object sender, EventArgs e)
{
    /* 执行解密操作 */
    DESSecurity des = new DESSecurity();
```

```
    this.txtDecode.Text =
      des.DESDecrypt(this.txtEncode.Text, this.txtDecodeKey.Text);
}
```

加密的时候，将“原文内容”文本框中的内容根据加密密钥进行加密，并把加密结果显示在“加密结果”文本框中；在解密的过程中我们将“加密结果”文本框中的内容进行解密，并把解密结果显示在“解密结果”文本框中。

4. 运行结果

运行项目，访问 DES.aspx 页面。

在“原文内容”文本框中输入“对称加密，也叫密钥加密。指加密和解密使用相同密钥的加密算法。”，在下面的“密钥”文本框中输入一个 8 字节的密钥“abcdefgh”，单击“加密”按钮，“加密结果”文本框中会显示出加密过的内容。

在“加密结果”下面的“密钥”文本框中同样输入密钥“abcdefgh”，单击“解密”按钮，“解密结果”文本框中会显示出解密后的内容，如图 12-3 所示。

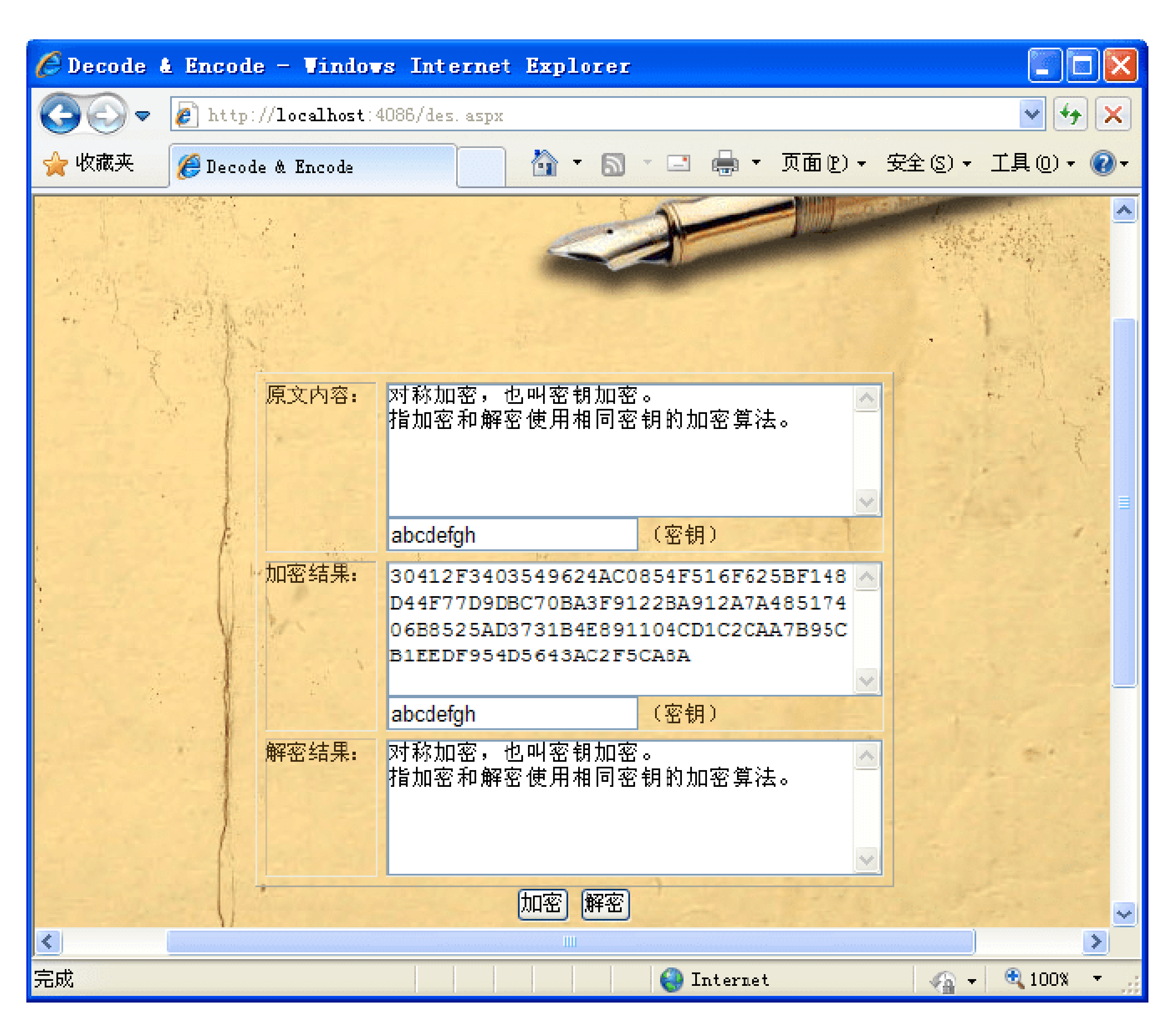


图 12-3　DES加密和解密

5. 实例分析



源码解析：

本实例的核心在 DESSecurity 类中。加密和解密的过程都需要将被加密或解密的文本转换成二进制数组，并写到加密流 CryptoStream 中，执行加密流 CryptoStream 对象的 FlushFinalBlock 方法进行加密，最后获取加密流中的二进制数组，并将其转换为字符串返回。

因为加密的密文是 16 进制的，所以需要手动编写代码进行二进制字符与字符串之间的转换操作。

12.1.3　不对称加密

前面也说过了，对称加密中，加密和解密双方使用的是同一个密钥。这样就导致在网络传输的时候，一旦密钥被有不良意图的人截获，我们传递的数据将不再保密。

不对称加密在对称加密的基础上对密钥进行优化，该加密方式使用一对匹配的公钥和私钥。加密和解密使用不同的密钥，这样相对于对称加密算法，不对称加密在安全性上有很大的进步。

 视频教学：光盘/videos/12/RSA.avi　　　　长度：8 分钟

1. 基础知识——不对称加密RSA

不对称加密算法的基本原理是：数据接收方生成一对公钥和私钥，并把公钥公开，数据发送方用公钥对数据进行加密，并发送给数据接收方，数据接收方收到数据后用私钥对数据进行解密，如图 12-4 所示。

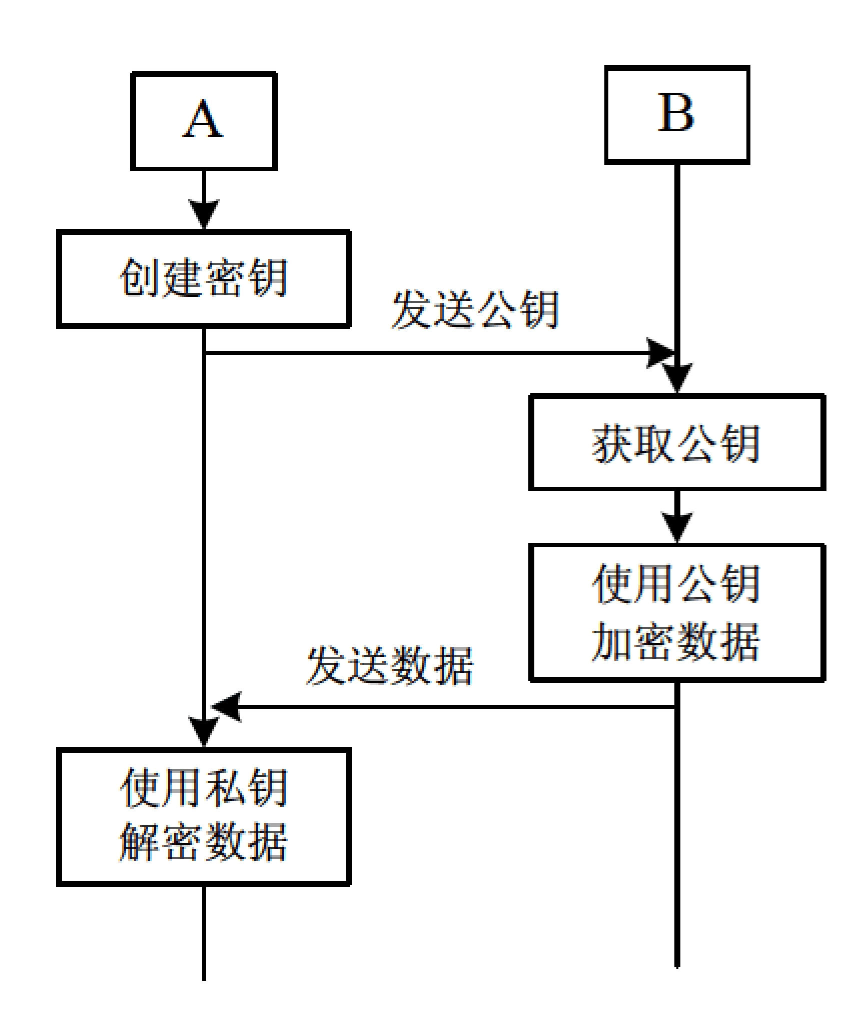


图 12-4　不对称算法的工作原理

广泛应用的不对称加密算法有 RSA 算法和美国国家标准局提出的 DSA 算法。以不对称加密算法为基础的加密技术应用非常广泛。



不对称加密算法的保密性比较好，它消除了最终用户交换密钥的需要，但相对于对称加密算法，它加密和解密花费时间长、速度慢，不适合于对文件加密而只适用于对少量数据进行加密。

ASP.NET 提供了一个 RSACryptoServiceProvider 类，可以创建一个使用不对称加密的 RSA 算法的加密服务提供者。

该类提供了一些方法，常用的方法如下。

- ToXmlString：该方法返回 XML 格式的字符型密钥。需要有一个 bool 型参数，参数为 true，返回的是私钥，为 false 返回的是公钥。

- FromXmlString：设置 XML 格式的字符型密钥，在进行加密或解密的时候使用该密钥进行操作。
- Encrypt：执行加密操作。
- Decrypt：执行解密操作。

2. 实例描述

下面我们也写一个使用 RSA 算法进行加密和解密的工具。

运行该工具时，首先应获取一对匹配的公钥和私钥，然后输入原文内容，可以使用公钥进行加密，然后可以使用私钥进行解密。

3. 实例应用

【例 12-3】不对称加密。

(1) 打开上节我们使用的项目。

(2) 创建一个封装密钥的实体类 RSAKey，代码如下：

```
public class RSAKey
{
    public string PrivateKey { get; set; }
    public string PublicKey { get; set; }
}
```

(3) 创建一个使用 RSA 算法进行加密的工具类，取名为 RSASecurity。该类有 3 个方法，一个创建密钥，一个执行加密，一个执行解密。代码如下：

```
public class RSASecurity
{
    public static RSAKey GetKeys()
    {
        RSAKey rsaKey = new RSAKey();    //声明一个 RSAKey 对象
        /* 创建一个 RSA 加密服务提供者 */
        RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
        rsaKey.PrivateKey = rsa.ToXmlString(true);    //创建私钥
        rsaKey.PublicKey = rsa.ToXmlString(false);    //创建公钥
        return rsaKey;    //返回结果
    }

    public static string Encode(string code, string key)
    {
        /* 将文本转换成 byte 数组 */
        byte []source = Encoding.Default.GetBytes(code);
        byte []ciphertext;    //密文 byte 数组
        /* 创建一个 RSA 加密服务提供者 */
        RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
        rsa.FromXmlString(key);    //设置公钥
        ciphertext = rsa.Encrypt(source, false);    //加密，得到 byte 数组

        /* 对字符数组进行转码 */
```

```
        StringBuilder sb = new StringBuilder();
        foreach (byte b in ciphertext)
        {
            sb.AppendFormat("{0:X2}", b);
        }
        return sb.ToString();    //返回结果
    }

    public static string Decode(string code, string key)
    {
        /* 将文本转换成 byte 数组 */
        byte []ciphertext = new byte[code.Length/2];
        for (int x=0; x<code.Length/2; x++)
        {
            int i = (Convert.ToInt32(code.Substring(x * 2, 2), 16));
            ciphertext[x] = (byte)i;
        }
        byte []source;    //原文 byte 数组
        /* 创建一个 RSA 加密服务提供者 */
        RSACryptoServiceProvider rsa = new RSACryptoServiceProvider();
        rsa.FromXmlString(key);    //设置私钥
        source = rsa.Decrypt(ciphertext, false);    //解密，得到 byte 数组

        return Encoding.Default.GetString(source);    //返回结果
    }
}
```

(4) 创建一个 Web 窗体，取名为 RSA.aspx。

(5) 在页面上添加 5 个文本框，分别用于输入或展示公钥、私钥、原文内容、加密结果、解密结果等，再添加 3 个命令按钮，分别用于执行获取密钥、加密、解密等操作。代码如下：

```
<table width="550" cellpadding="5" cellspacing="5" border="1">
<tr>
<td colspan="2">
公钥：<asp:TextBox ID="txtPublic" runat="server" Height="50" Width="500"
  TextMode="MultiLine"></asp:TextBox><br />
私钥：<asp:TextBox ID="txtPrivate" runat="server" Height="50" Width="500"
  TextMode="MultiLine"></asp:TextBox>
</td>
</tr>
<tr>
<td width="80">原文内容：</td>
<td>
<asp:TextBox ID="txtSource" runat="server" Height="60" Width="400"
  TextMode="MultiLine"></asp:TextBox><br />
</td>
</tr>
<tr>
<td>加密结果：</td>
```

```
<td>
<asp:TextBox ID="txtEncode" runat="server" Height="60" Width="400"
  TextMode="MultiLine"></asp:TextBox><br />
</td>
</tr>
<tr>
<td>解密结果：</td>
<td>
<asp:TextBox ID="txtDecode" runat="server" Height="60" Width="300"
  TextMode="MultiLine"></asp:TextBox>
</td>
</tr>
</table>
<asp:Button ID="btnGetKey" runat="server" Text="获取密钥"
  onclick="btnGetKey Click" /> 
<asp:Button ID="btnEncode" runat="server" Text="加密"
  onclick="btnEncode_Click" /> 
<asp:Button ID="btnDecode" runat="server" Text="解密"
  onclick="btnDecode_Click" />
```

(6) 在页面后台添加命令按钮的事件处理程序，代码如下：

```
protected void btnGetKey_Click(object sender, EventArgs e)
{
    RSAKey key = RSASecurity.GetKeys();    //获取密钥
    this.txtPrivate.Text = key.PrivateKey;
    this.txtPublic.Text = key.PublicKey;
}

protected void btnEncode_Click(object sender, EventArgs e)
{
    /* 把原文内容使用公钥进行加密 */
    this.txtEncode.Text =
      RSASecurity.Encode(this.txtSource.Text, this.txtPublic.Text);
}

protected void btnDecode_Click(object sender, EventArgs e)
{
    /* 把原文内容使用私钥进行解密 */
    this.txtDecode.Text =
      RSASecurity.Decode(this.txtEncode.Text, this.txtPrivate.Text);
}
```

程序首先生成一对密钥，并添加到页面控件中。然后在执行加密的时候，用生成的公钥对“原文内容”文本框中的内容进行加密。在进行解密的时候，用生成的私钥对“加密结果”文本框中的内容进行解密。

4. 运行结果

运行项目，访问 RSA.aspx 页面。

单击“获取密钥”按钮，在“公钥”和“私钥”文本框中显示出来生成的密钥。

在原文内容中输入一行文本，比如“床前明月光光光”，单击“加密”按钮，在“加密结果”文本框中显示出加密后的内容。

单击“解密”按钮，程序将加密过的文本进行解密并展示在“解密结果”文本框中。如图 12-5 所示。

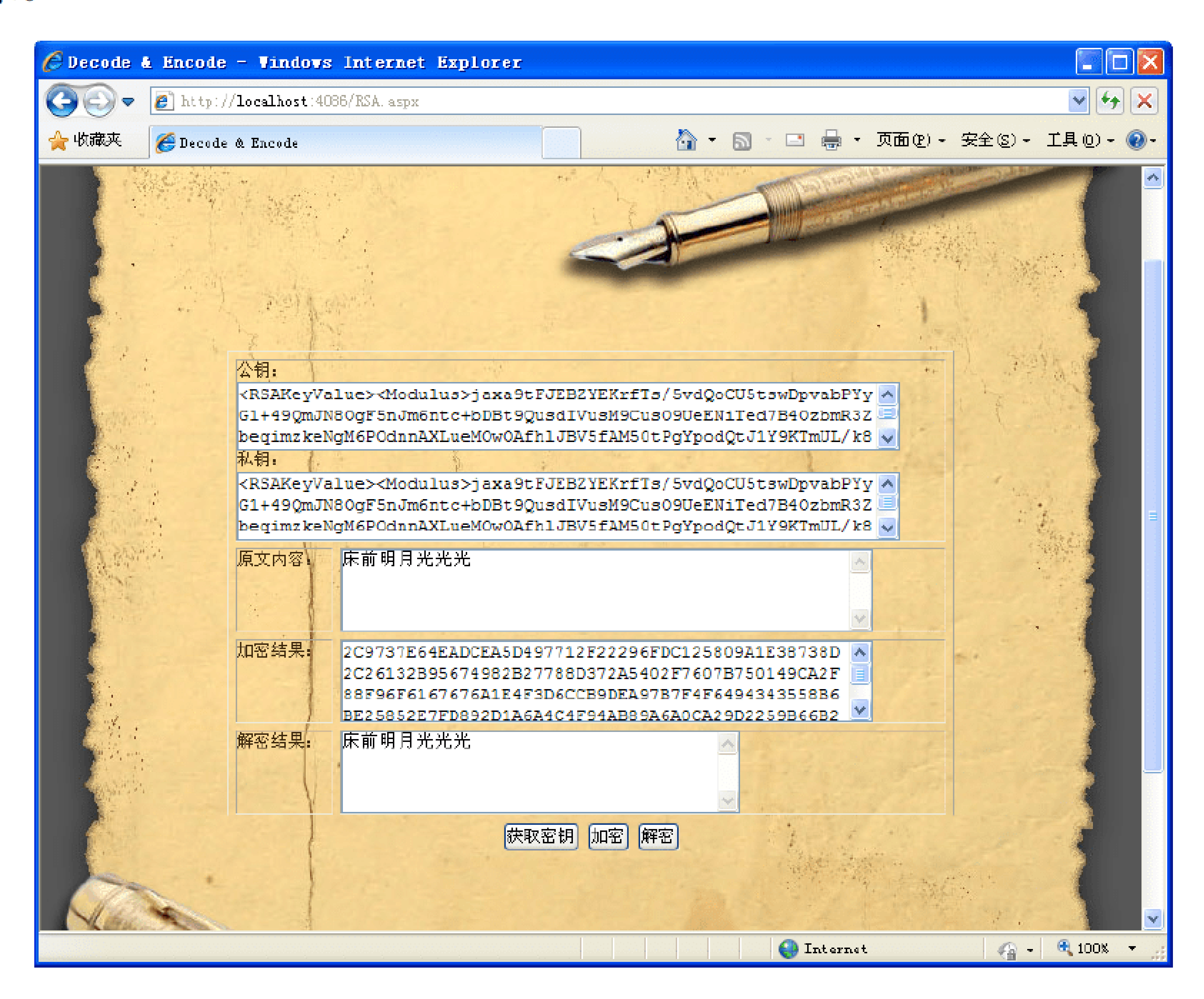


图 12-5　RSA加密解密

5. 实例分析



源码解析：

实例首先使用 RSACryptoServiceProvider 实例的 ToXmlString 方法获取一对配匹的公钥和私钥。在进行加密或解密的时候需要首先使用 RSACryptoServiceProvider 实例的 FromXmlString 方法设置密钥，再调用 Encrypt 方法或者 Decrypt 方法进行加密或者解密。

当然，这里的密文也需要对其进行编码转换。

12.1.4　使用散列保护数据

有时候我们保护一些机密的数据，只需要单向加密。比如在数据库中保存用户的密码。

对用户账号的密码，我们没有查看的必要，使用它只是用来在登录的时候验证一下。而且为了保证用户的隐私，我们把用户密码视为最高机密，所以任何人都不能获取明文的密码，这样这个账户的密码就只可能由设置密码的那个人知道了。

散列就是这样一个单向的加密算法。



视频教学：光盘/videos/12/MD5.avi　　长度：6 分钟

1. 基础知识——散列码MD5 加密

散列，英文是 Hash，这种编码叫做 Hash Code，译过来就是散列码或者直译为哈希码。

散列就是把任意一段数据经过某种算法生成一段唯一的固定长度的数据，也叫做摘要。就是说把任意长度的输入通过散列算法变换成固定长度的输出，该输出就是散列值。这种转换是一种压缩映射，也就是说，散列值的空间通常远小于输入的空间，不同的输入可能会散列成相同的输出，而不可能从散列值来唯一地确定输入值。

再简单地说，散列就是一种将任意长度的消息压缩到某一固定长度的消息摘要的方法。

Hash 是主要应用于信息安全领域中加密算法。目前最为广泛使用的 Hash 算法有 MD5 和 SHA1。



MD：Message Digest(信息摘要)
SHA：Secure Hash Algorithm(安全散列算法)

MD5(RFC 1321)是 Rivest 于 1991 年对 MD4 的改进版本。它的输入仍以 512 位分组，其输出是 4 个 32 位字的级联，与 MD4 相同。MD5 比 MD4 来得复杂，并且速度较之要慢一点，但更安全，在抗分析和抗差分方面表现更好。

SHA1 是由 NIST NSA 设计为同 DSA 一起使用的，它对长度小于 2^{64} 的输入，产生长度为 160bit 的散列值，因此抗穷举性更好。

SHA-1 设计时基于和 MD4 相同原理，并模仿了该算法。

这里我们以 MD5 为例讲一下 Hash Code。

ASP.NET 的 System.Security.Cryptography 命名空间提供了一个 MD5CryptoServiceProvider 类，该类可以创建一个使用 MD5 算法的加密服务提供对象。

该类不需要使用密钥就能对数据进行加密。但是这种加密是“不可逆转”的。



这里的“不可逆转”只是比较理想的情况，是官方宣称的，但实际上 MD5 已经被破解。

该算法使用比较简单，直接调用该类实例的 ComputeHash 方法即可。当然，这里操作的仍然是 byte[]类型的数据。

2. 实例描述

本实例中我们开发一个使用 MD5 算法加密的工具。功能比较简单，只需要输入被加密的内容，执行加密后直接返回加密过的文本内容。

3. 实例应用

【例 12-4】使用散列保护数据。

(1) 打开前面使用的项目。

(2) 创建一个 Web 窗体，命名为 MD5.aspx。

(3) 在页面加入两个文本框控件，一个输入原文内容，一个展示加密过的内容，再加入一个命令按钮，执行加密任务。

代码如下：

```
<table width="500" cellpadding="5" cellspacing="5" border="1">
<tr>
<td width="80">原文内容：</td>
<td>
<asp:TextBox ID="txtSource" runat="server" Height="60" Width="400"
  TextMode="MultiLine"></asp:TextBox><br />
</td>
</tr>
<tr>
<td>加密结果：</td>
<td>
<asp:TextBox ID="txtEncode" runat="server" Height="60" Width="400"
TextMode="MultiLine"></asp:TextBox><br />
</td>
</tr>
</table>
<asp:Button ID="btnEncode" runat="server" Text="加密"
  onclick="btnEncode_Click" />
```

(4) 在页面后台代码中加入命令按钮的单击事件处理程序。代码如下：

```
protected void btnEncode Click(object sender, EventArgs e)
{
    /* 获取原文内容的 byte 数组 */
    byte []sourceCode = Encoding.Default.GetBytes(this.txtSource.Text);
    byte []targetCode;    //声明用于获取目标内容的 byte 数组

    /* 创建一个 MD5 加密服务提供者 */
    MD5CryptoServiceProvider md5 = new MD5CryptoServiceProvider();
    targetCode = md5.ComputeHash(sourceCode);    //执行加密

    /* 对字符数组进行转码 */
    StringBuilder sb = new StringBuilder();
    foreach (byte b in targetCode)
    {
        sb.AppendFormat("{0:X2}", b);
    }

    this.txtEncode.Text = sb.ToString();    //返回数据
}
```

4. 运行结果

运行项目，访问 MD5.aspx 页面。

在“原文内容”文本框里输入要加密的内容，例如“admin”，单击“加密”按钮执行加密操作，返回加密结果，如图 12-6 所示。

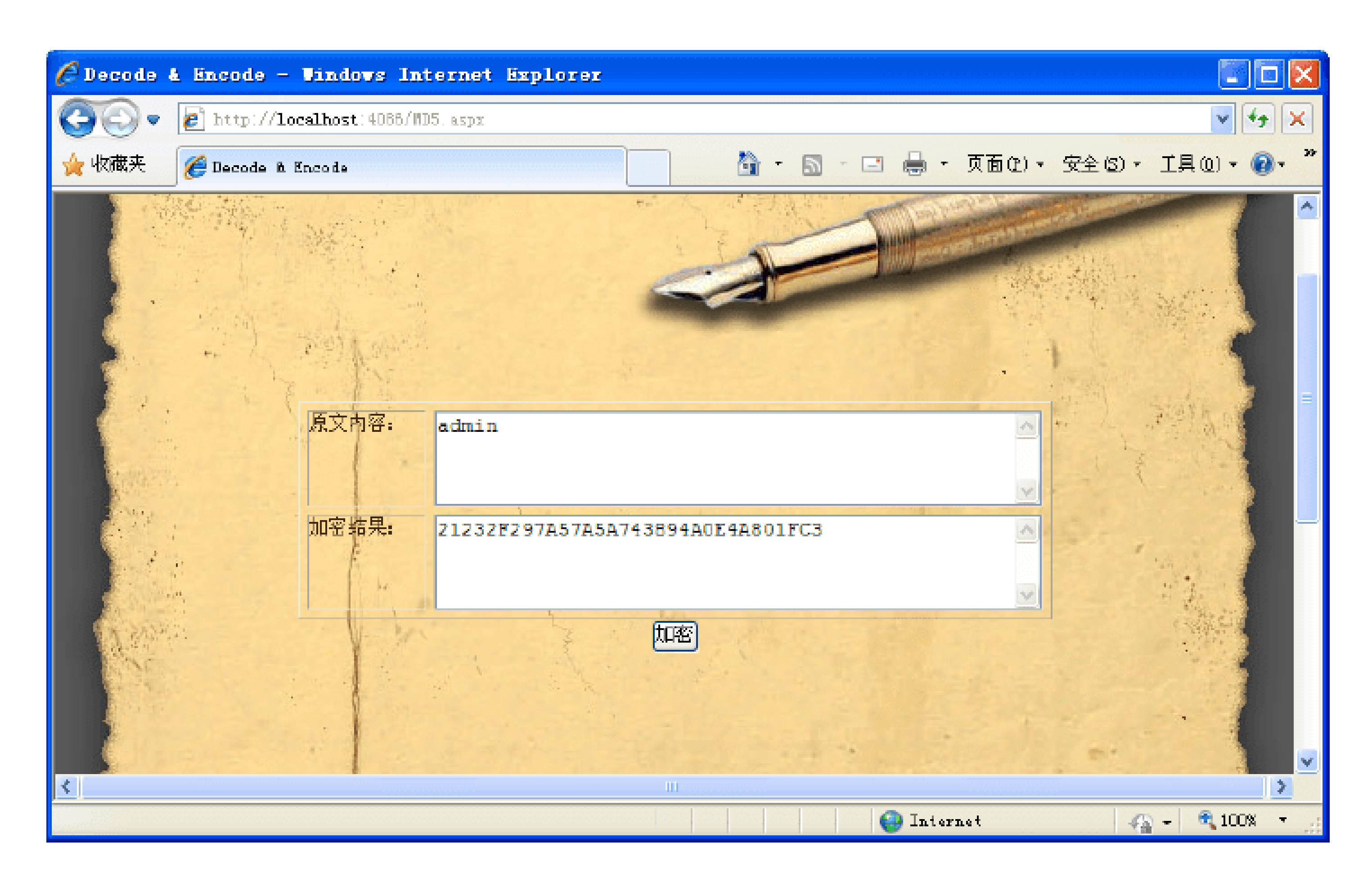


图 12-6　MD5 算法加密

5. 实例分析



源码解析：

实例使用 MD5 算法对文本内容进行 Hash 加密。首先获取用户输入的内容，并将其转换为 byte 数组，然后使用 MD5CryptoServiceProvider 类实例的 ComputeHash 方法对 byte 数组进行加密，得到一个加密过的 byte 数组，对其进行编码，即可得到加密过的文本内容。

12.1.5　SQL注入

SQL 注入，这个名字听着好抽象啊。我们听过注入水的猪肉、注入糖的西瓜、SQL 注入什么呢？

我们来看一个小例子。

以前在处理用户登录验证的时候，通常是怎么做的呢？特别是一些 ASP 程序员的习惯。我们常这样拼接一个 SQL 语句：

```
String sql = "select * from [Users] where [Username]='" + username
  + "' and [Password]='" + password + "'";
```

上面的语句能得到类似这样的一个 SQL 语句：

```
select * from [Users] where [Username]='admin' and [Password]='123'
```

执行了这条 SQL 语句，我们就完成了对数据库带条件的查询操作，如果条件满足，返回满足信息的记录，如果不满足，返回一个空列表。

上面的变量 username 和 password 都是从用户界面文本框中获取到的值。但假如一个了解或者通过非法手段知道了数据库结构的人在用户名文本框中输入了下面这句话：

```
a' or 1=1;insert [Users] values('joker','123',1) --
```

结果我们拼接成的 SQL 语句就成这样了：

```
select * from [Users] where [Username]=' a' or 1=1;insert [Users]
values('joker','123',1) --' and [Password]='123'
```

来分析一下，这个 SQL 语句可以被断为三个部分：

```
select * from [Users] where [Username]=' a' or 1=1;
insert [Users] values('joker','123',1)
--' and [Password]='123'
```

第一行以分号结尾，是一个完整的语句，意思是如果用户名为 a，或者 1 等于 1，就返回该记录。“或者 1 等于 1”，这不净废话，肯定返回所有的数据，该用户也肯定登录成功了。

第二行更坏，又直接在用户表里插入一个用户，用户名 joker，密码 123，权限为 1。以后就可以使用这个用户正常地出入系统了。

第三行成为注释语句，本来我们要进行密码验证的，却简单地用两个减号注释掉就完了。

其实，这个攻击者已经很仁慈了，他没有把那条 insert 语句换成 drop 语句就已经非常够意思了。

这样，冥冥之中，我们的系统就被一个使用注入手段的不法分子给捣毁了。

当然，知道了攻击手段，我们就能拿出相应的解决方案。在 ASP.NET 里，我们可以使用以下几种手段防止 SQL 注入给我们带来损失：

- 过滤单引号。SQL 语句中如果需要向数据库提交一个单引号符号，我们可以使用单引号对单引号进行转义，就是在内容里用两个单引号来代表一个单引号符号。
- 不要使用拼接的 SQL 语句，应该使用参数化的 SQL 语句。
- 对数据库用户的权限做详细的划分。比如创建、修改或删除表结构的权限设为更高一级权限的用户，把执行查询和增加、更新、删除信息的用户区分开。
- 一些机密内容例行加密后存放，这样即使系统崩溃也不至于丢失机密数据。

比如上面我们提到的问题，最优雅的办法就是使用第二种方法。使用 ASP.NET 提供的 Command 对象的参数机制。

代码如下：

```
string sql =
"select * from [Users] where [Username] = @Username and [Password] = @Password";
SqlCommand cmd = new SqlCommand(sql, conn);    //conn 为 Sqlconnection 对象
cmd.Parameters.Add(new SqlParameter("@Username", this.TextBox1.Text));
cmd.Parameters.Add(new SqlParameter("@Password", this.TextBox2.Text));
```

在这里，我们对查询语句设置了用户名和密码两个参数，如需要，还可添加任意多个。

当然，上面说的只是这一种注入方法的解决方案，有时候我们可能会遇到一些意外情况。能不能根据具体的问题分析、构造出合适的、巧妙的 SQL 语句，是 SQL 注入安全的关键。

12.1.6　图片防盗链

有些摄影爱好者照了一些漂亮的风景照片，或者一些商家发布了一些精美的产品图片，或者一些广告公司发布了一些创意作品……

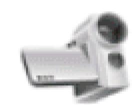
当他们把这些图片上传到网上以后，不久后发现，许多大大小小的论坛、博客都大量引用

这些图片。更可气的是，有些还都直接地引用图片路径，在不增加他们网站点击量的前提下对服务器性能造成大量的剽窃。

这种可以称为“可耻”的剽窃行为有一个专业的学术名词，叫“盗链”。

当然，网络如此之大，对其数据进行监管根本是不可能的事情。所以，我们需要在数据源头(我们的站点上)对这种行为进行巧妙的防止。

下面来研究一下图片的防盗链方法。



视频教学：光盘/videos/12/HttpHandler.avi　　长度：9 分钟

1. 基础知识——HttpHandler图片防盗链

图片防盗链的方法很多，各有优劣。我们先来了解一下。

- 禁用右键菜单：这种方法能且仅能防止一般的初级 Web 使用者，任何稍懂 HTML 的用户查看源代码后将一览无余。
- 使用空白图片覆盖真实图片：这只是一种障眼法，这种情况下用户在图片上单击的时候下载和查看到的只是那个透明的图片，但是如果用户查看源代码，对 HTML 稍微一分析，就能得到一个真实的图片地址。
- 预先生成带水印的图片：生成水印的方法很有效。原理是在图片上盖上一个我们网站的印章，无论谁在哪里使用，我们不再眼睁睁地吃亏了。因为他们在盗链我们图片的同时，无形中为我们做了广告，两全其美。不过预先生成的水印图片对图片源文件进行了修改，即使是我们自己，也得不到图片本来的面貌了，对我们来说也不是最好的解决方案。
- 访问时生成水印：也是用水印，不过是在对该图片请求时动态加的水印，在我们的服务器硬盘存储的是没有加水印的图片。不过这种方法每次访问都要加一次水印，对服务器处理性能是个不小的考验。
- 使用 Flash 来显示图片：这种方法基本上是防盗链的最有效的方法，无论使用什么方法，用户都无法得到图片的实际路径(除非他进入站点管理后台)。但是这种方法也有一点不是太好，我们展示一个小小的图片还得另外加载一个 Flash 控件，也有点浪费性能。
- 杜绝本站以外的地址访问图片：使用程序判断用户上一次的请求路径，如果不是本站，就意味着图片被其他站点引用了，我们可以拒绝显示图片。

上面所说的这些方法只能在一定程度上防止图片的散布，但是作为网站的访问者，用户有一招非常无敌的方法，是我们在服务器端完全无法制止的。

那就是——截图。

对这种特殊方法我们不做考虑。

下面我们来介绍一下最后一种防盗链的方法：禁止本站以外的地址访问图片(其他的方法有兴趣的读者自己研究)。

要进行图片防盗链控制，就得了解一下 ASP.NET 的底层机制了。因为 Web 服务器为了节省开销，默认时总是在遇到客户端请求的时候把图片文件、静态页面之类的资源型的文件直接发送给用户，并不经过 ASP.NET 的过滤。所以我们得想办法让 ASP.NET 能够捕获对图片文件的请求，并编写程序对其进行处理。

ASP.NET 提供了一个名为“管道”的机制，ASPNET 下的每一个请求都要通过这个管道请求资源，然后对请求的响应也通过这个管道返回客户端。

这个管道可以分为三个部分：请求的入口 HttpRuntime、请求的过滤 HttpModuler、请求的处理 HttpHandler。Web 服务器把请求交给 HttpRuntime，HttpRuntime 把请求放入管道，管道里的 HttpModuler 对请求进行过滤，最后请求到达 HttpHandler，由 HttpHandler 对请求做最终的处理。如图 12-7 所示。

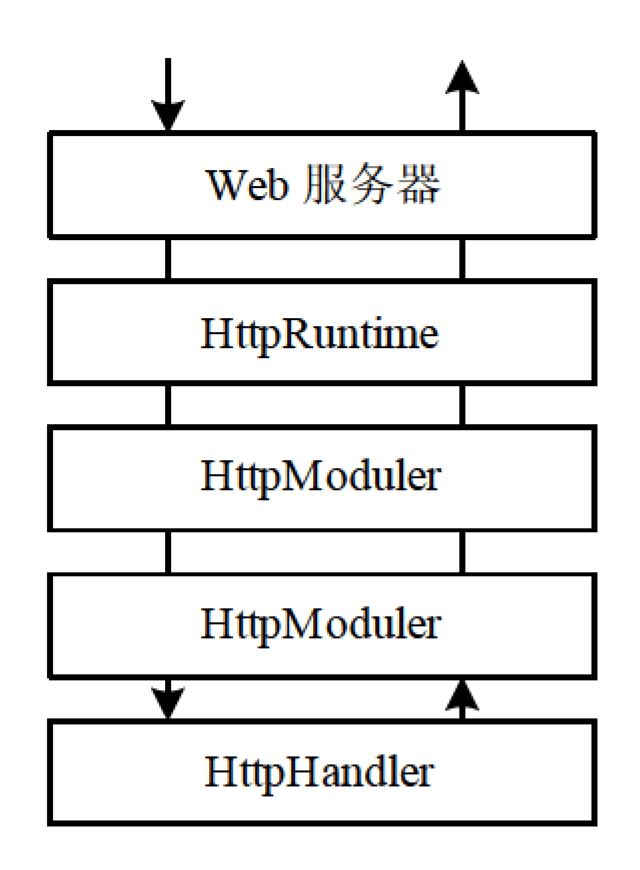


图 12-7　ASP.NET管道示意



这里 HttpModuler 画了两个，其实可以有多个。它只是个过滤器，就像一条路上可以设许多检查站一样，可以多个 HttpModuler 共存。

到这里可能有读者会郁闷了，我们访问的是一个 WebForm，最终处理的是页面，这里怎么说是一个 HttpHandler 呢？

打开我们 ASP.NET 目录下的 web.config 文件(默认在 C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG 目录下)，我们就会发现在 httpHandlers 下面有如下一个节点：

```
<add path="*.aspx" verb="*" type="System.Web.UI.PageHandlerFactory"
  validate="true"/>
```

该节点将所有扩展名是 aspx 的请求使用 System.Web.UI.PageHandlerFactory 处理。所以看似我们在地址栏访问的是 aspx 页面，其实进行处理的是一个 HttpHandler。

上面节点中的 path 即是被处理的路径，可使用通配符和扩展名来指定一类请求；verb 指的是请求方式(POST、GET 等)；type 就是要处理的 HttpHandler 类的全路径了。知道这些就够了。

这里我们就自定义一个 HttpHandler 来接收所有对 JPG 图片文件的请求，并进行防盗链处理。自定义 HttpHandler 需要实现 IHttpHandler 接口。IHttpHandler 接口定义在 System.Web 命名空间中。

该接口有一个公有方法 ProcessRequest 和一个属性 IsReusable。

ProcessRequest 方法用来处理用户请求。通过该方法的 HttpContext 类型的参数我们可以访问 Application、Request、Response、Server 等服务器对象。

IsReusable 属性用来标识该 HttpHandler 对象是否可以重复使用。如果是，则整个应用程序只会创建一个该 HttpHandler 类的对象，如果否，应用程序会对每一个请求创建一个该

HttpHandler 类的对象。

另外，也可以在该应用程序的 web.config 文件里配置 HttpHandler。在应用程序的 web.config 文件里的 system.web 节点下有一个 httpHandlers 节点，可在这里配置自定义的 HttpHandler。

2. 实例描述

以前开发过一个个人版的博客，里面经常会发布一些漂亮的图片。

不过因为访问量不大，只买了一个很便宜的虚拟主机。这个虚拟主机对连接数和月流量都有限制。

为了防止博客上的图片被别人盗链，不得已，把自己上传的图片全都加了防盗链功能。如果有站外的链接访问这些资源，全都给他们返回一个很难看的图片。

3. 实例应用

【例 12-5】图片防盗链。

(1) 打开前面我们使用的项目。

(2) 新建一个“upload”文件夹，存放两张图片：普通的图片 lian.jpg 和默认的防盗链图片 default.jpg。

(3) 打开日志列表首页 index.html，在内容区添加一个 img 标签，引用 upload 文件夹下的 lian.jpg 图片。代码如下：

```
<img alt="" src="upload/lian.jpg" class="c_pic"/>
```

(4) 新建一个“一般处理程序”，命名为 JpgHandler。

(5) 修改 ProcessRequest 方法，代码如下：

```
public void ProcessRequest(HttpContext context)
{
    /*
     * 程序获取当前请求的主机名和当次请求的前面一次请求的主机名
     * 进行比较后如果相等，说明是合法请求，如果不等，说明是非法请求
     * (因为如果正常访问，在请求图片以前肯定会请求一下所访问的页面)
     */

    HttpRequest request = context.Request;    //获得 Request 对象
    string serverPath = context.Server.MapPath("~/");    //获取当前应用程序根路径
    string curPath =
      string.Empty, prePath = string.Empty;    //声明当前路径和上次请求路径

    curPath = request.Url.Host;    //获取当前 URL 的主机名
    /* 如果前一次请求不为空，获取前一次请求的主机名 */
    if (request.UrlReferrer != null) prePath = request.UrlReferrer.Host;

    if (curPath == prePath)   //比较，如果相等，返回请求的图片，如果不等，返回默认图片
    {
        string imgPath = request.Url.AbsolutePath;  //获取请求的绝对路径，图片的 URL
        imgPath = imgPath.Substring(imgPath.LastIndexOf('/'));  //取图片的文件名
        context.Response.WriteFile(serverPath + "upload/" + imgPath);
        return;
```

```
    }
    else
    {
        context.Response.WriteFile(serverPath + "upload/default.jpg");
    }
}
```

(6) 在 web.config 里配置该 Handler。打开 web.config 文件，在 system.web 下的 httpHandlers 节点下添加如下节点：

```
<add verb="*" path="upload/*.jpg" type="Web1.JpgHandler" />
```

这里指定对 upload 目录下的所有扩展名是 jpg 的文件的请求都交给 Web1 命名空间下的 JpgHandler 处理。

4. 运行结果

运行程序，访问 index.html 页面。结果如图 12-8 所示。



图 12-8　Blog首页

我们看到，图片正常显示。再开一个浏览器窗口，直接访问这个图片的路径：upload 目录下的 lian.jpg。访问结果如图 12-9 所示。

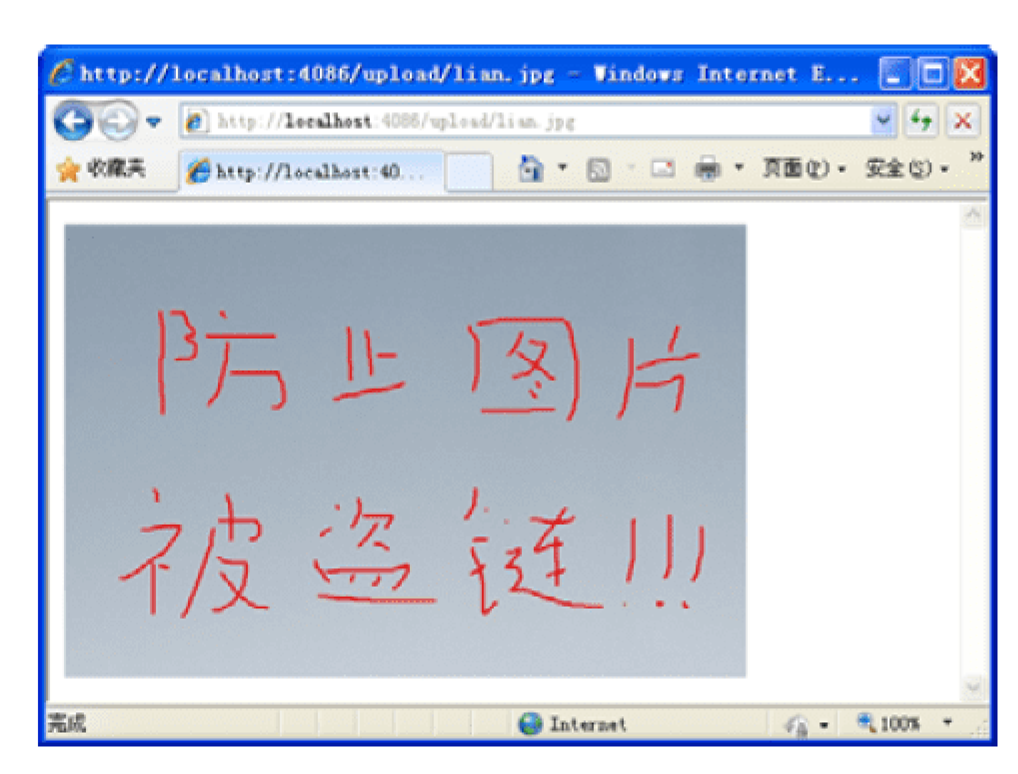


图 12-9　图片防盗链

5. 实例分析

源码解析：

实例利用 ASP.NET 的 HttpHandler 机制对 upload 目录下的 JPG 文件进行处理。

在处理的时候判断如果当前请求(对图片的请求)的前一次请求的主机名等于当前站点的主机名，说明是本站点对图片的引用，否则说明是其他站点的引用或者浏览器直接下载。这里对该次请求返回一个默认的图片。

12.2 程序编码优化

计算机的“快”确实给人类的生活带来了很大的便利，为我们节约了不少时间。

但是，随着计算机程序功能越来越强大，对硬件的要求也越来越高。虽然硬件也在一代代地升级，但是高性能的程序一直也是软件开发人员的追求。

打开以前我们写的程序，多多少少都有一些“糟糕的”代码。现在看看简直惨不忍睹。

接下来我们就来了解一下程序代码的优化方案，对我们以前的代码进行一些改造。

12.2.1 集合操作优化

集合就是存在于内存中的一组具有某种相同特征的数据，最典型的就是数组。

数组是内存中一块连续的存储区，访问速度很快。但缺点是长度固定，不方便进行插入、追加、删除元素的操作，也难以使用其他的数据结构来存储数据(比如键-值对)。如需要更高的集合类型，ASP.NET 在 System.Collections 和 System.Collections.Specialized 这两个命名空间下提供了支持这些操作的类型。

选择使用合适的集合可以使代码变得更加简洁和高效。不同的集合类型执行特定任务的能力也不同，相应地，在性能上来说也有很大的差别。

对集合操作性能优化最典型的就有数组和 ArrayList 的选用和泛型的使用两个重点。下面我们来了解一下。

1. 数组和ArrayList的选用

以最常用的集合——数组为例。普通的数组声明方式如下：

```
int []score = new int[5];
```

这种数组的特点是效率高，但是长度固定，执行插入、删除之类的操作不太方便。

.NET Framework 在 System.Collections 命名空间下面提供了一个 ArrayList 集合类，称为动态数组。特点是长度自由，使用方便。但是性能上来说要比普通数组稍差一点。下面我们来分析一下原因。

普通的数组只是在内存中为引用分配一个定长的内存空间，并将变量名引用到数组的第一个元素上，如图 12-10 所示。

score → | 0 | 1 | 2 | 3 | 4 |

图 12-10　数组的内存分配

在普通数组中，我们如果要对其进行插入、删除等操作，需要考虑元素序列问题，还有数组长度问题，使用时有诸多不便。

ArrayList 的本质是对普通数组的封装，其具体实现还是普通的数组，它在内存中分配的空间也和如图 12-10 所示的一样。

之所以它比普通数组执行编辑操作的时候方便，是因为它在执行编辑操作的时候，如果遇到原数组长度不够或其他之类的情况，将自动地创建并引用一个新的数组来满足需要。也就是说，在我们不知道的情况下，其实系统已经自动创建了一个新的数组，并对原来的数组进行完全拷贝，再执行新的操作。

比如，我们用以下代码来创建一个新的 ArrayList：

```
ArrayList al = new ArrayList();
```

这时我们若调用该数组的 Capacity 属性获取数组容量，得到的结果是 0。说明我们这里只是一个空数组。但是，对其进行一次 Add 操作的时候，我们会发现，数组的容量直接变成了 4，而使用 Count 属性得到的实际大小却只是 1。我们继续对其进行 Add 操作，到 Count 属性为 5 的时候，Capacity 属性的值已经是 8。长度变了，说明数组换了，这个时候虽然还是像以前一样执行了一次 Add 操作，但是，系统自动创建了一个长度为 8 的数组，将原来长度为 4 的数组里的内容拷贝到这个数组，再执行 Add 操作。

这时，若数组长度需要超过 8，系统会自动创建一个长度为 16 的数组，再进行一次拷贝。也就是说默认情况下 ArrayList 以 2 的指数进行扩张。

试想一下，如果我们需要一个长度为 70 的数组，为图方便使用了 ArrayList，并且是以 ArrayList 的默认构造方法创建的对象。那么，光在执行 Add 方法的时候，内存中就会创建 6 个数组，而且 5 个都没有用了。

多大的性能浪费！

所以这里建议，使用尽量直接使用普通数组。需要使用 ArrayList 的时候，使用 ArrayList 类的带参构造方法创建一个基本足够长度的 ArrayList。如下所示：

```
ArrayList student = new ArrayList(70);
```

总体来说，普通的数组和 ArrayList 相比，各有优劣，使用的时候要尽量权衡利弊。

2. 泛型的使用

泛型是.NET 2.0 中新推出的一种强大的功能。我们知道通过泛型可以定义类型安全的数据结构，这能够显著提高代码的性能并得到更高质量的代码，而且使代码的易读性也得到提高。

本节，我们只来了解一下泛型机制对程序性能的贡献。

我们知道，在使用诸如 ArrayList、Hashtable 之类的传统的集合的时候，对集合里数据的操作都需要使用类型转换，而数据类型的转换通常是非常浪费性能的，特别是可能还要涉及到装箱和拆箱操作，批量的操作更是对运行环境性能的极大糟踏。

下面我们来看一个实例：

```
static void Main()
{
    int length = 100 * 10000;    //执行次数，这里设置为一百万次
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    /* 使用 ArrayList 进行数据存取 */
    ArrayList al = new ArrayList(length);
    for (int i=0; i<length; i++)
    {
        al.Add(i);
        int num = (int)al[i];
    }
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    /* 使用泛型进行数据存取 */
    List<int> il = new List<int>(length);
    for (int i=0; i<length; i++)
    {
        il.Add(i);
        int num = il[i];
    }
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    Console.ReadLine();    //程序暂停，等待键入回车结束
}
```

程序声明了一个 ArrayList 和一个泛型的数组集合 List<int>，分别对两个集合进行 100 万次数据存取操作。并在每次执行前后打印当前时间的计时周期数。

程序执行结果如下：

```
634200611335937500
634200611337500000
634200611337812500
```



DateTime 对象的 Ticks 属性返回的 long 型数值单位是 100 纳秒，即 1/10000 毫秒。

我们发现第一次没用泛型的时候，对基本数据类型 int 值操作需要进行拆箱和装箱，耗费了 150 多毫秒，而第二次使用泛型的时候，仅仅使用了 30 多毫秒。

所以，在应用程序开发时使用集合对象的时候应尽可能地使用泛型集合来存取数据。

12.2.2 字符串连接优化

在.NET Framework 中，字符串对象在内存中是永远固定不变的。因为其固定不变的特性，对其修改就成了问题。

比如两个字符串连接，.NET Framework 采用的是创建一个新的字符串，将原来的两个字符串 Copy 到新的字符串中，然后将组成的新字符串赋给指定的引用。这个时候，内存里其实已经创建了 3 个字符串对象。如图 12-11 所示。

Web程序员成功之路

h	e	l	+	l	o	

str → h e l l o

图 12-11　字符串连接

但是，运算符加号(+)在处理字符串类型的数据的时候，也是按数学运算一样是从左到右依次执行。在这种情况下，如果是 3 个或更多的字符串相连，将会出现何种情况？

比如 3 个字符串 a、b、c 相连，内存中将会生成 a、b、c、ab、abc 等 5 个字符串；如果是 4 个字符串 a、b、c、d 相连，内存中将会生成 a、b、c、d、ab、abc、abcd 等 7 个字符串；如果是 5 个字符串相连，将会生成 9 个字符串……

依次类推，需要连接的字符串越多，在内存中生成的字符串也就会越多。由此得出生成的字符串个数的计算公式是 $2 \times n-1$。而除了初始的几个字符串和最终生成的字符串以外，其余的字符串都是废物，对我们毫无用处而且浪费性能。

.NET Framework 提供了一个字符串建造者类 StringBuilder，可以避免大量的性能浪费。

StringBuilder 是在内存中创建一个缓冲区，先将第一个字符串放入缓冲区，如果有 Append 方法需要执行，就直接在原字符串后面进行追加。因为不再创建一大堆字符串，所以性能上有质的提高。

我们通过一个例子，来了解一下使用 StringBuilder 和使用普通的字符串相加在性能上的差别。代码如下：

```
static void Main()
{
    int length = 20000;    //执行次数，这里设置两万次
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    StringBuilder sb = new StringBuilder("0");
    for (int i=0; i<length; i++)
    {
        sb.Append(i);
    }
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    string str = "0";
    for (int i=0; i<length; i++)
    {
        str += i;
    }
    Console.WriteLine(DateTime.Now.Ticks);    //打印时间的计时周期数
    Console.ReadLine();    //程序暂停，等待键入回车结束
}
```

程序分别用两种方法对字符串“0”连接从 0 到 20000 的 20001 个数字，结果相差很大：

```
634200722333125000
634200722333281250
634200722365625000
```

使用 StringBuilder 连接只需要十几毫秒，而用加号对字符串连接就需要 3000 多毫秒。足足 3 秒多钟。

如此可以看到 StringBuilder 的足够优势，在字符串连接的时候该如何选择就不言而明了吧。

12.2.3　类型转换优化

通常在进行类型转换的时候会因为某种原因可能会抛出异常。而我们知道抛出异常是非常浪费资源的。所以这里类型转换的优化主要是对转换中可能存在的异常的避免。

程序运行时很多时候都需要将字符串类型转换为数值类型，比如用户输入的年龄值。在转换之前，我们往往对用户的输入内容不可知，所以转换很可能会失败。

而通常我们对其转换使用的强制转换和使用类型的 Parse 方法转换都会在类型转换失败的时候抛出异常。

在.NET 2.0 以上的版本中，为每个基本类型提供了 TryParse 方法。以 int 型为例，该方法的语法如下：

```
int.TryParse(s, out result);
```

参数 s 是被转换的字符串，输出参数 result 是接收转换结果的 int 型变量。该方法的返回值是 bool 类型，true 代表转换成功，false 代表转换失败。

该语句执行后如果转换成功，result 的值为转换后的值，如果转换失败，result 的值为 0。

当然，对于引用类型，也有强制转换和非强制转换。但是系统并没有给每一个类提供一个 TryParse 方法，我们在对引用类型的数据进行类型转换的时候需要使用 as 关键字。该方法的语法如下：

```
ArrayList a = o as ArrayList;
```

上面的代码中，对象 o 是 object 类型的对象。

在这里进行转换的两个类型一般是父类和子类的关系，否则转换后的结果为 null，而不是实际的对象。

另外，类型转换还通常出现在值类型和引用类型之间，也称之为“装箱”和“拆箱”。Object 好比一个箱子，将值类型转换为引用类型(即对象)是“装箱”，反之是“拆箱”。

“装箱”操作和“拆箱”操作都需要大量的计算，没有必要的装箱和拆箱操作对程序性能也是极大的浪费。

12.3　ASP.NET网站的性能优化

Web 站点终究是要运行到网络上的。在网络上的程序由于网络环境、服务器性能，以及程序设计等各方面的原因，速度会变得很慢。

像网络环境、服务器性能这些原因，我们作为程序员是无法进行控制的，所以程序员的责任就是把我们的网站优化到最高的性能就好了。

有了前面的知识，虽然我们基本上已经可以实现一般常见的功能了，但是程序性能却是程

序健壮性的必要条件。

下面我们就来了解一下如何优化我们的 ASP.NET 站点。

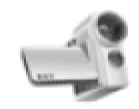
12.3.1 实现ASP.NET探针

现在的网络条件还不能满足人人都可以随意架设 Web 服务器的需求。所以，现在网络上大部分个人用户或小型企业用户都是靠租用一些互联网数据中心(Internet Data Center，IDC)的虚拟主机来架设自己的网站。

而有些 IDC 纯粹就是奸商，号称的“高配置、超稳定”的服务器经常无法访问，而且在可以访问的时候速度还很慢。

而且我们也不具备去现场查看服务器硬件的条件，只能在远程使用一些手段进行核实。

这就要用到“探针”。

 **视频教学：**光盘/videos/12/System.avi　　长度：6 分钟

1. 基础知识——系统底层操作

“探针”其实就是一段获取服务器信息的程序。具体地说，所谓 ASP.NET 探针，其实就是使用.NET 类库获取服务器的软、硬件配置信息，通过页面展示给浏览器，供用户查看的一段程序。

服务器的域名、IP、端口之类的信息可以直接使用 Request 对象或者 Server 对象获取，其他的信息就可以调用.NET 的其他类库进行查询。

System 命名空间下提供了一个 Environment 静态类，可以获取环境和系统的一些信息。

其中 OSVersion 属性可以获取当前操作系统的版本，TickCount 属性可以获取当前服务器已运行时间，Version 属性可以获取当前通用运行库的版本号。

Version 属性隶属 System.Version 类，可以详细地获取 Major、Minor、Build、Revision 等版本的信息。

另外 GetEnvironmentVariable 方法还可以根据传入的参数获取其他的环境信息。比如 CPU 个数、CPU 类型等。

使用 System.IO 命名空间下的 Directory 类的 GetLogicalDrives()方法可以获取当前服务器上的逻辑驱动器编号列表。

使用 Microsoft.Win32 命名空间下的 Registry 类可以获取指定注册表节点上的值。

另外，使用 System.Diagnostics 下的 Process 类可以获取当前程序运行时的信息，通过该类的 GetCurrentProcess()方法可以获取一个 Process 对象，进而得到当前程序运行的详细信息，如内存使用率、CPU 使用状态等。

使用 System 命名空间下的静态类 GC 还可以对垃圾回收机制进行管理和查看。

2. 实例描述

网上已经有很多探针程序可以直接下载来使用。不过这一节我们要通过编写代码实现一个自己的 ASP.NET 探针程序。

3. 实例应用

【例 12-6】实现 ASP.NET 探针。

(1) 新建一个 Web 项目。

(2) 打开 Default.aspx 页面，向页面中添加一些 Label 标签，用于显示服务查询得到的服务器信息(代码省略)。

(3) 我们需要在页面初始化的时候做查询，并把得到的服务器信息填充到页面上。打开 Default.aspx 页面的后台代码，在 Page_Load 方法里添加代码如下：

```
protected void Page Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        //使用服务器变量获取服务器名称
        lbDomain.Text = Request.ServerVariables["SERVER_NAME"].ToString();
        lbIp.Text =
          Request.ServerVariables["LOCAl_ADDR"];    //使用服务器变量获取本机地址
        lbPort.Text = Request
          .ServerVariables["Server_Port"].ToString();  //使用服务器变量获取端口
        lbOperat.Text = Environment.OSVersion.ToString();  //获取系统版本
        /* 获取逻辑驱动器编号 */
        foreach (string drivesName in Directory.GetLogicalDrives())
        {
            lbLogicDriver.Text += drivesName;
        }
        lbIISVer.Text = Request.ServerVariables["Server_SoftWare"]
          .ToString();    //使用服务器变量获取服务器软件及版本
        /* 获取 IE 浏览器版本 */
        RegistryKey key = Registry.LocalMachine
          .OpenSubKey(@"SOFTWARE\Microsoft\Internet Explorer\Version Vector");
        lbIEVer.Text = key.GetValue("IE", "未检测到").ToString();
        lbServerLastStartToNow.Text = ((Environment.TickCount/0x3e8)/ 60)
          .ToString() + "分钟";    //获取服务器运行时间
        lbCpuNum.Text = Environment.GetEnvironmentVariable(
          "NUMBER_OF_PROCESSORS").ToString();    //获取 CPU 个数
        lbCpuType.Text = Environment.GetEnvironmentVariable(
          "PROCESSOR_IDENTIFIER").ToString();    //获取 CPU 类型
        lbMemory.Text = ((Double)Process.GetCurrentProcess().WorkingSet64
          /(1024 * 1024)).ToString("N2") + "M";    //获取进程内存使用量
        lbMemoryPro.Text = ((Double)GC.GetTotalMemory(false)
          /(1024 * 1024)).ToString("N2") + "M";    //获取本应用程序使用的内存量
        lbPhPath.Text = Request.PhysicalApplicationPath;  //获取应用程序绝对路径
        lbTimeOut.Text =
          Server.ScriptTimeout.ToString() + "毫秒";    //获取脚本超时时间
        /* 获取 ASP.NET 版本 */
        lbAspnetVer.Text = string.Concat(new object[] {
            Environment.Version.Major, ".", Environment.Version.Minor, ".",
            Environment.Version.Build, ".", Environment.Version.Revision
```

```
        });
        lbCpuNet.Text = ((TimeSpan)Process.GetCurrentProcess()
          .TotalProcessorTime).TotalSeconds.ToString();//获取 ASP.NET 的 CPU 时间
        lbSessionNum.Text =
          Session.Contents.Count.ToString();  //获取 Session 个数
    }
}
```

(4)　保存所有代码。

4. 运行结果

运行项目，访问 Default.aspx 页面。运行结果如图 12-12 所示。



图 12-12　ASP.NET探针程序

5. 实例分析



源码解析：

本实例使用.NET Framework 提供的类库来访问系统参数，得到系统配置信息和环境变量。本实例只是简单地列举了一部分服务器配置信息，.NET Framework 提供了许多强大的功能，可以对系统进行更深入的查看或设置。

本实例的目的不在于能够编写一个探针程序，而是使我们能够调用类库获得更多的系统信息，以对我们程序运行的环境有一个更深入的了解。

12.3.2　使用Server.Transfer()方法

在前面的章节中，我们知道在 ASP.NET 中，页面程序进行页面之间的切换有两种方式：Response 对象的 Redirect 方法和 Server 对象的 Transfer 方法。

Response 对象的 Redirect 方法是直接向浏览器返回一个地址，由浏览器再次请求服务器返回来的地址获取响应信息，原理如图 12-13 所示。

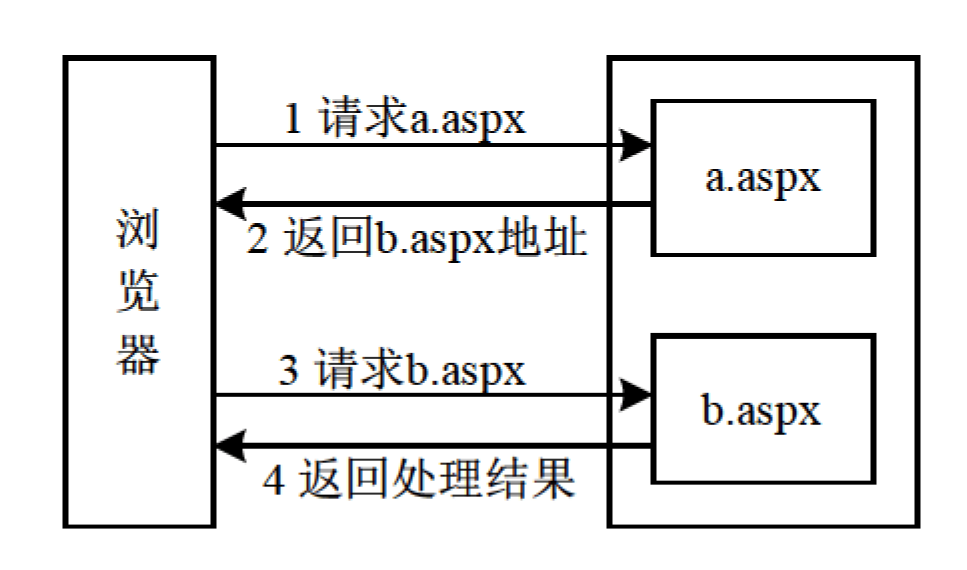


图 12-13　Response对象的Redirect方法重定向请求的原理

Server 对象的 Transfer 方法是直接将请求转发给当前应用程序的页面，像一个代理一样代替客户端请求新的页面，在获得请求后返回给客户端，原理如图 12-14 所示。

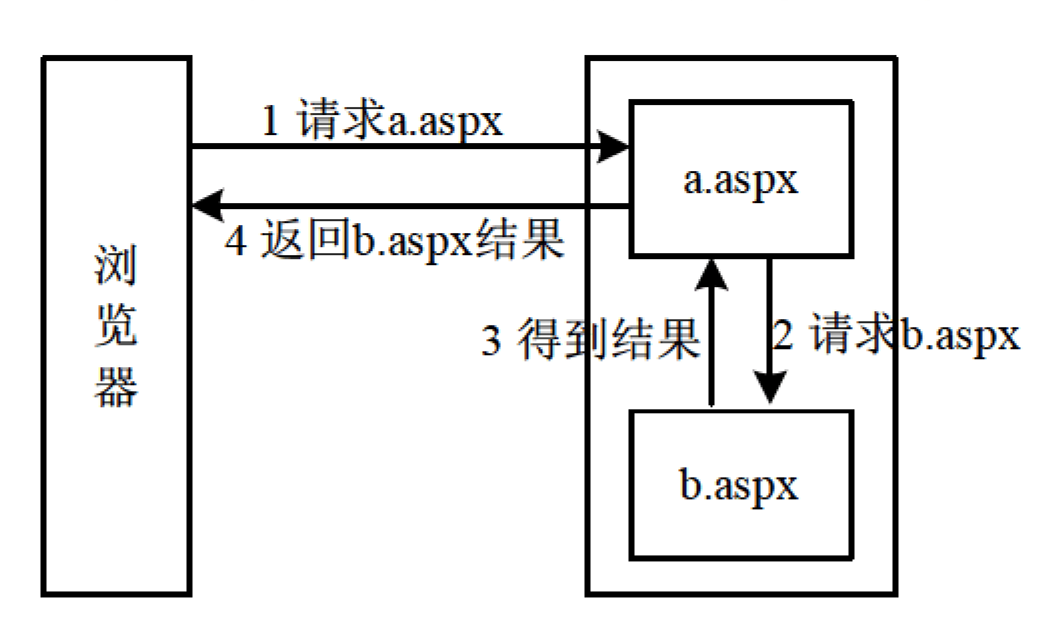


图 12-14　Server对象的Transfer方法转发请求的原理

这样，使用 Server 对象的 Transfer 方法实现同一应用程序下不同页面间的重定向可以避免不必要的客户端页面重定向。使用 Server 对象的 Transfer 方法相对于 Response 对象的 Redirect 方法来说，可以让客户端少执行一次请求，性能稍微要高一点。

并且 Server 对象的 Transfer 方法具有允许目标页从源页中读取控件值和公共属性值的优点。由于调用了这个方法之后浏览器上不会反映更改后的页的信息，因此它也适合以隐藏 URL 的形式向用户呈现页面，不过如果用户点击了浏览器上的“后退”按钮或者刷新页面，有可能导致意外情况。



不过这里要注意，Server 对象的 Transfer 方法只能在当前站点里进行使用。在跨站点的系统中，就必需使用 Response 对象的 Redirect 方法。

12.3.3　合理使用ViewState

前面章节中我们讲 Page 对象的时候讲过页面(Page)的生命周期。在页面中，有一个对象几乎贯穿整个生命周期，那就是 ViewState。

ViewState 可以说是 ASP.NET 的 WebFrom 机制的运行核心。ViewState 对象可以在整个页面运行周期内进行状态保持，所以在页面生存周期内暂存一些数据时使用 ViewState 对象是非常方便的。

但是，ViewState 对象不能随意乱用。

前面我们讲过，ASP.NET 最终会把 ViewState 对象进行某种编码操作后放置到页面顶部的

一个隐藏域中，如图 12-15 所示。



图 12-15　页面源文件

也就是说，在 ViewState 对象中保存着的所有的状态信息全部都会进行编码，存入隐藏域，发回客户端。待下次请求提交到服务器后，再对隐藏域里的数据进行解码，然后才能取出使用数据。

在整个过程中，我们存储的数据将要经过编码、在网络上传输、再传输、解码等过程才能使用，而这些过程都是非常消耗服务器和网络的性能的。

首先编码和解码这两个过程中需要建立大量的对象，对字符串进行组合、拆分、解析，比较浪费服务器的内存、CPU 等资源；而传输过程就更不可预料了，上面短短的一小段代码，就有差不多七八百个字符，近 1KB 了。这么一段数据，每一次请求和响应都会在服务器和客户端之间进行传递，对网络的浪费可想而知。

鉴于此，在 ASP.NET 的 WebForm 中尽可能不用或者少用 ViewState 进行数据存储，以减轻服务器和网络的压力。

12.3.4　禁用调试模式

在我们新建过一个 Web 项目，第一次单击 VS 菜单栏上的“启用调试”按钮运行的时候，会弹出一个“未启用调试”对话框，如图 12-16 所示。

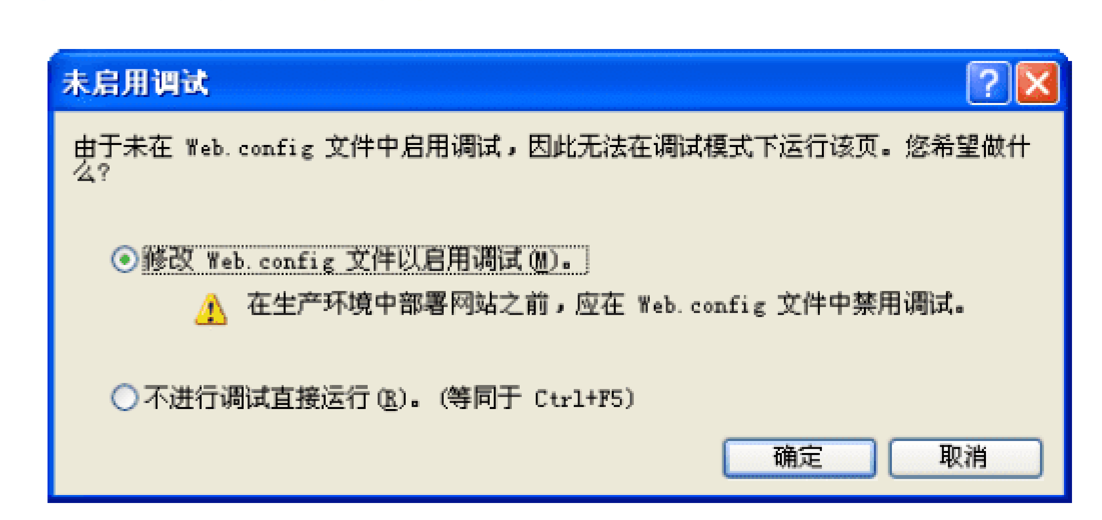


图 12-16　未启用调试

在这里如果选中“修改 Web.config 文件以启用调试”单选按钮，并单击“确定”按钮以后，系统会自动修改 Web.config 文件，使程序运行并进入调试模式。

ASP.NET 支持在便于开发人员进行故障排除的程序调试模式下运行程序。这样在程序运行的时候，开发人员就可以通过设置断点的方式对程序运行的关键部位进行一步步的运行和调试。在这种状态下，ASP.NET 启用了调试器，密切监视和控制应用程序执行的额外信息。

以这种方式执行程序对开发人员来说非常方便，这样，应用程序在运行的时候就不再需要调试，而且可以获得尽可能高的性能。然而调式模式对应用程序的性能影响是非常大的，所以在应用程序正式发布的时候，应尽量关掉应用程序的调试功能。

打开 Web.config 文件，发现 system.web 节点下面有如下配置：

```
<!--
    设置 compilation debug="true" 可将调试符号插入已编译的页面中。但由于这会
    影响性能，因此只在开发过程中将此值设置为 true。
-->
<compilation debug="true">
     ...  //配置代码省略
</compilation>
```

注释代码已经很清楚地说明，程序编译的时候启用调式模式，而且程序会因此而影响性能，所以建议只在开发过程中设置该值为 true。

所以，我们若想提高程序运行性能，需要在这里将 compilation 节点的 debug 属性设置为 false 来关掉调式模式即可。

12.4　数据访问优化

应用程序最终是要处理数据的，在大数据量的系统中，数据访问的优化才是整个应用程序优化的重中之重。特别是数据库和应用程序往往并不在一台服务器上存放，对数据库服务器资源的浪费简直就是不可容忍的过失行为。

下面我们来了解一下对数据库操作的优化。

12.4.1　数据库连接对象使用优化

刚才说过，对数据库服务器的资源浪费是非常不能容忍的行为。所以，我们在对数据库操作的时候，要尽量节约数据库服务器的资源，比如数据库连接。

对于数据库连接，一个“亘古不变”的原则就是：尽可能晚地打开连接，并且尽可能早地关闭连接。

因为数据库连接就是数据库服务器的资源，这个程序多占用一秒钟，可能系统就会因此迟钝一秒钟。



现在一些数据库服务器提供商那里的数据库服务器对一般用户都限制连接数，有的竟限制为一个数据库 1 个连接。

我们来看看下面这段代码：

```
SqlConnection conn = new SqlConnection(connStr);    //创建连接
conn.Open();    //打开连接
string sql = "select * from [Users] where [Age] > @Age";  //SQL 字符串
SqlParameter para = new SqlParameter("@Age", age);  //创建 SQL 参数对象
SqlCommand cmd = new SqlCommand(sql,conn);    //创建 Command 对象
cmd.Parameters.Add(para);    //添加参数
IList<UserInfo> users = new List<UserInfo>();  //创建用户集合
SqlDataReader reader = cmd.ExecuteReader();    //获得一个 DataReader 对象
/* 读取数据 */
while (reader.Read())
{
    users.Add(new UserInfo() {
        Username = reader.GetString(1),
        Password = reader.GetString(2)
    });
}
/* 绑定数据 */
this.GridView1.DataSource = users;
this.GridView1.DataBind();
conn.Close();  //关闭连接
```

程序从创建完连接对象开始，打开连接，一直到页面绑定完数据，才关闭连接对象。对数据库服务器性能产生极大的浪费。

首先，像创建 SQL 语句、创建参数、创建 Command 对象、添加参数、创建用户集合、为页面控件绑定数据等操作都不需要使用数据库连接，这个时候打开数据库连接着实是对数据库连接的极大浪费。

我们稍做修改，如下所示：

```
SqlConnection conn = new SqlConnection(connStr);    //创建连接
string sql = "select * from [Users] where [Age] > @Age";  //SQL 字符串
SqlParameter para = new SqlParameter("@Age", age);  //创建 SQL 参数对象
SqlCommand cmd = new SqlCommand(sql, conn);    //创建 Command 对象
cmd.Parameters.Add(para);    //添加参数
IList<UserInfo> users = new List<UserInfo>();  //创建用户集合
conn.Open();    //打开连接
SqlDataReader reader = cmd.ExecuteReader();    //获得一个 DataReader 对象
/* 读取数据 */
while (reader.Read())
{
    users.Add(new UserInfo() {
        Username = reader.GetString(1),
        Password = reader.GetString(2)
    });
}
conn.Close();  //关闭连接
/* 绑定数据 */
this.GridView1.DataSource = users;
this.GridView1.DataBind();
```

整个程序其实也就在执行 SqlCommand 对象的 ExecuteReader 方法对 SqlDataReader 对象进行遍历的时候需要数据库连接。所以，我们只在进行这两个操作的时候打开连接，执行完立刻关闭，尽可能地节约连接资源。

12.4.2　优化SQL语句

其实在整个 ASP.NET 程序性能优化的所有方法中，对 SQL 语句的优化是整个优化工作的重中之重。在程序开发中十分优秀的业务逻辑实现被一句 SQL 命令拖死的情况不在少数。因为在大数据量的系统中，对数据库操作就显得格外重要，一丝一毫的差异都会积累成大的性能差别。

拿所有程序中最常使用的数据分页功能来说吧，分页功能的 SQL 语句有 N 种写法，而且每一种都有差别。下面随便举几个例子，来看一下不同的语句间有多大差异。

先说一下测试数据库。这里使用的是安装 VS2008 时默认的开发版的 SQL Server 2005，随便建一个数据库。有一个数据表 table2，表里有一个自增主键列 id，有一个 varchar(50)的普通字段 ddddd。表里有随意添加了 10485760 行数据。硬件环境就是公司里的工作用机，配置一般。

首先，来看一个最简单的方法——数据全读出来，然后使用 GridView 展示、分页：

```
select * from table2
```

测试了两次，分别是 176 秒和 173 秒。将近 3 分钟，估计再绑定到 GridView 上，人都快急疯了。不过数据量小的话也没什么，百十条记录的差异是感觉不出来的。

接下来看另一种：使用 SQL Server 2005 的特有函数 row_number()来建立一个索引列，使用 between 关键字筛选指定范围的记录。SQL 语句如下：

```
select * from (
select *, row_number() over (order by [id] asc) as rowNum from table2
) as table1 where rowNum between 10000001 and 10000005
```

用这种方法在查询第 101 到 105 条记录的时候，用了 0 秒就完成了。在查询第 10000001 到 10000005 条记录的时候，测试了 5 次，结果分别是 12 秒、13 秒、13 秒、12 秒、12 秒。效果还行，不算很差，而且网上也有很多人推崇这种用法。

下面再来看一种方法。

比如我们要查第 101 到 105 条记录，我们可以查出 100 条以前的，排除掉，再在剩余的记录中找前 5 条，就是我们要的列表。

代码如下：

```
select top 5 * from table2 where id not in (select top 10000000 id from table2)
```

用这种方法在查询第 101 到 105 条记录的时候，也是用了 0 秒就完成了。在查询第 10000001 到 10000005 条记录的时候，测试了 5 次，结果分别是 14 秒、16 秒、14 秒、14 秒、14 秒。这个是作者在学习时第一次要解决分页问题想到的办法，基本上还行，效果稍微差了一点点。

再看一种方法。

还可以用这一种思路进行分页：比如要取 101 到 105 之间的记录，我们可以先查出前 105

条，反转一下顺序，取前 5 条，就得到所要的记录了。不过这个时候是倒序的，再顺序排一下就可以了。

代码如下：

```
select * from (
    select top 5 * from (
        select top 10000005 * from table2 order by id
    ) as t1 order by id desc
)as t2 order by id asc
```

对这种方式进行测试。在查询第 101 到 105 条记录的时候，还是用了 0 秒就完成了。在查询第 10000001 到 10000005 条记录的时候，也测试了 5 次，结果全都是 8 秒。效率似乎要比前几种都要高。

到此为止，我们就不再过多地介绍了。

从这几种方法中，我们就可以看到很大的性能差异。一条好的 SQL 语句的选择，可能让系统每次执行都节省数秒的时间。相对于优化程序代码来说，对 SQL 语句的优化在时间和风险成本方面都很低。

不过 SQL 太简单，入门要求不高，易学，但难以精通，而且一个问题可以有很多种解决方案。所以能否写出优雅、易读、高效的 SQL 语句，也是衡量一个程序员水平高低的因素。

12.5　对计算器模块进行单元测试

现在几乎所有的程序语言都在向“面向对象”的设计思想转型，相应地，使用这些语言开发出来的程序都具备了“面向对象”的一个最大的特点，那就是“封装”。

封装使程序更加模块化，也使程序开发的分工更加容易。比如一个结合很严密的业务可以分给几个人实现不同的步骤。

但是，这样的分工相应地在代码测试环节方面可能就不太方便了。因为每个人写的进度不一样，往往我们写完一个功能模块却很难测试它的正确性。

对于这个问题，程序界的牛人们为大家提供了很多不错的解决方案。网上已有不少现成的单元测试工具可以使用。

使用在.NET 平台上的，名气最响的莫过于 NUnit 了。

 **视频教学**：光盘/videos/12/NUnit.avi　　　　长度：11 分钟

12.5.1　基础知识——NUnit单元测试

单元测试(也叫模块测试)，是开发人员编写的一小段代码，用于检测程序代码的一个很小的、很明确的功能模块是否正确。

在单元测试中，被测试的独立单元将在与程序其他相关部分相隔离的情况下进行测试。单元测试被作为一种无错编码的辅助手段运用在软件开发过程中。

比如汽车厂在造一辆汽车，造成之前对汽车车灯、刹车系统、音响系统、安全系统等模块进行单独的测试，就可以被称为单元测试。

单元测试是为了证明程序模块与我们当初设计的功能相一致。

NUnit 是.NET Framework 平台下的一个单元测试框架，是专门针对于.NET 来写的。

NUnit 框架使用方法比较简单，它只需要我们简单地编写一下对我们程序模块单元进行测试的类和方法，使用 NUnit 框架提供的属性对相应的测试类和方法进行标记，它就可以识别并进行测试。在 NUnit 框架中，客户属性是非常重要的。我们来介绍其中最常用的 3 种。

- TestFixture：该属性将一个类标记为包含测试的容器。使用该属性标记的类需要满足两个要求：必须是公有(Public)的，必须有默认的无参构造方法。因为 NUnit 框架将在程序外部的命名空间里使用默认无参构造方法声明该类的实例。
- SetUp：该属性将标记一个初始化常规资源的方法。被它标记的方法将在任何测试之间执行。它对应一个标记销毁资源方法的属性 TearDown。
- Test：该属性将一个类(标记 TestFixture 属性的类)里的方法确定为可测试的方法。而且，该方法必需是一个公有的无返回值的方法。

在 NUnit 框架中，使用 Assert(断言)进行验证。

Assert 是一个类，它提供了许多静态方法。常用的有 AreEqual、AreSame、Fail、Ignore、IsTrue、IsNull 等。

- AreEqual：对各种类型的数据进行比较，如果验证不成功，则抛出一个 NUnit.Framework.AssertionException 异常来提示测试失败
- AreSame：验证两个参数是否是引用一个相同的对象。如果否，则抛出一个 NUnit.Framework.AssertionException 异常。
- Fail：直接抛出一个 NUnit.Framework.AssertionException 异常。
- Ignore：直接抛出一个 NUnit.Framework.AssertionException 异常。这个在测试报告中被忽略。
- IsTrue：验证条件是否为真。若不为真，抛出一个 NUnit.Framework.AssertionException 异常，对应地还有一个 IsFalse 方法。
- IsNull：验证对象是否为空。若不为空，抛出一个 NUnit.Framework.AssertionException 异常，对应地还有一个 IsNotNull 方法。

掌握了这些东西，我们就可以使用 NUnit 框加对我们的程序进行单元测试了。

12.5.2 实例描述

以前写过一个简单的计算器类，这个计算器类是一个独立的类库，没有其他的应用程序界面，所以不知道它的功能是否准确。

本实例我们就使用 NUnit 框架对这个计算器类进行单元测试。

12.5.3 实例应用

【例 12-7】对计算器模块进行单元测试。

(1) 导入我们的计算器模块的类库 Unit12.Counter 所在的项目。

计算器类 Counter 的代码如下：

```
/// <summary>
/// 计算器
/// </summary>
public class Counter
{
    //加
    public int Add(int num1, int num2)
    {
        return num1 + num2;
    }

    //减
    public int Subtract(int num1, int num2)
    {
        return num1 - num2;
    }

    //乘
    public int Multiply(int num1, int num2)
    {
        return num1 * num2;
    }

    //除
    public int Divide(int num1, int num2)
    {
        return num1 / num2;
    }
}
```

该类实现了加、减、乘、除 4 个基本的功能。这里我们抽取加法和除法进行测试。

(2) 新建一个项目，用于测试。

(3) 对该项目添加对 unit.framework 类库的引用，如图 12-17 所示。

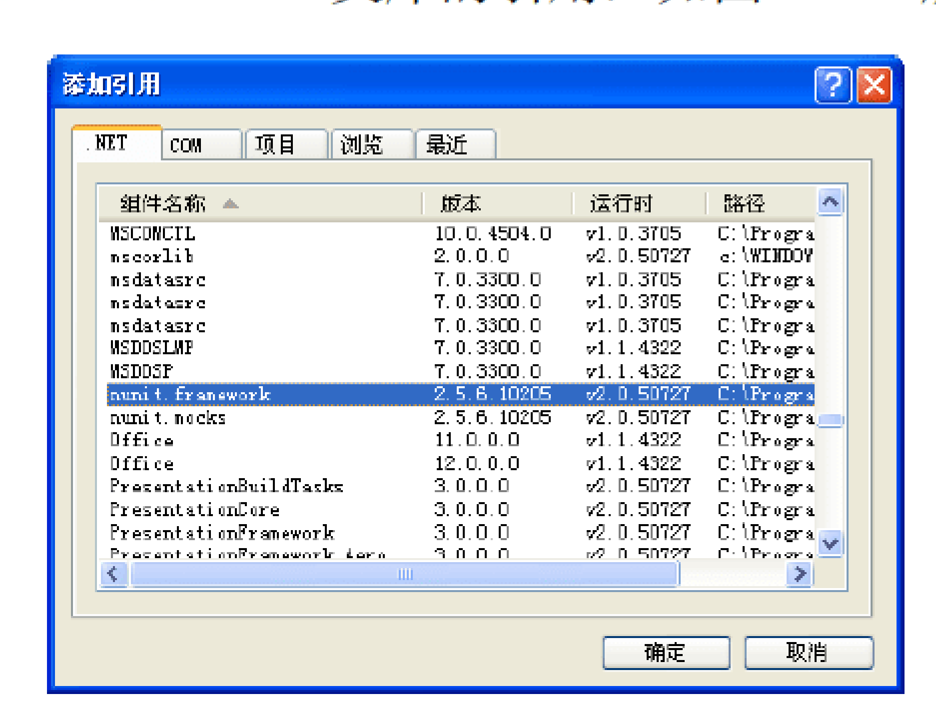


图 12-17　添加引用



如果读者的机器没有安装 NUnit 测试工具，这里可能没有这个类库选项。需要下载并安装 NUnit 测试工具以后即可在这里看到该项。

(4) 添加一个测试类 TestCounter，并在类里添加一个计算器字段和 3 个方法。

- counter 字段：用于下面方法中对计算器方法的测试。
- Init 方法：用于初始化计算器。
- TestAdd 方法：用于对计算器的加法方法进行测试。
- TestDivide 方法：用于对计算器的除法方法进行测试。

然后给该类添加 TestFixture 标记，对 Init 方法添加 SetUp 标记，对 TestAdd 方法和 TestDivide 方法添加 Test 标记。代码如下：

```
[TestFixture]
public class TestCounter
{
    Counter counter;

    [SetUp]
    public void Init()
    {
        counter = new Counter();
    }

    [Test]
    public void TestAdd()
    {
        int n1 = 10;
        int n2 = 20;
        int result = counter.Add(n1, n2);
        Assert.AreEqual(result, 30);
    }

    [Test]
    public void TestDivide()
    {
        int n1 = 2;
        int n2 = 0;
        int result = counter.Divide(n1, n2);
        Assert.AreEqual(result, 0);
    }
}
```

这里测试除法的时候设置的被除数为 0，测试的时候应该会在这里抛出异常，一会儿注意一下。

(5) 在“解决方案资源管理器”里右击测试项目，在弹出的快捷菜单中选择“属性”菜单命令，打开项目的属性窗口。

切换到“调试”选项卡，在“启动操作”选项组下面选中“启动外部程序”单选按钮，并

选择 NUnit 工具的路径(这里是默认路径 C:\Program Files\NUnit 2.5.6\bin\net-2.0\nunit.exe)，如图 12-18 所示。

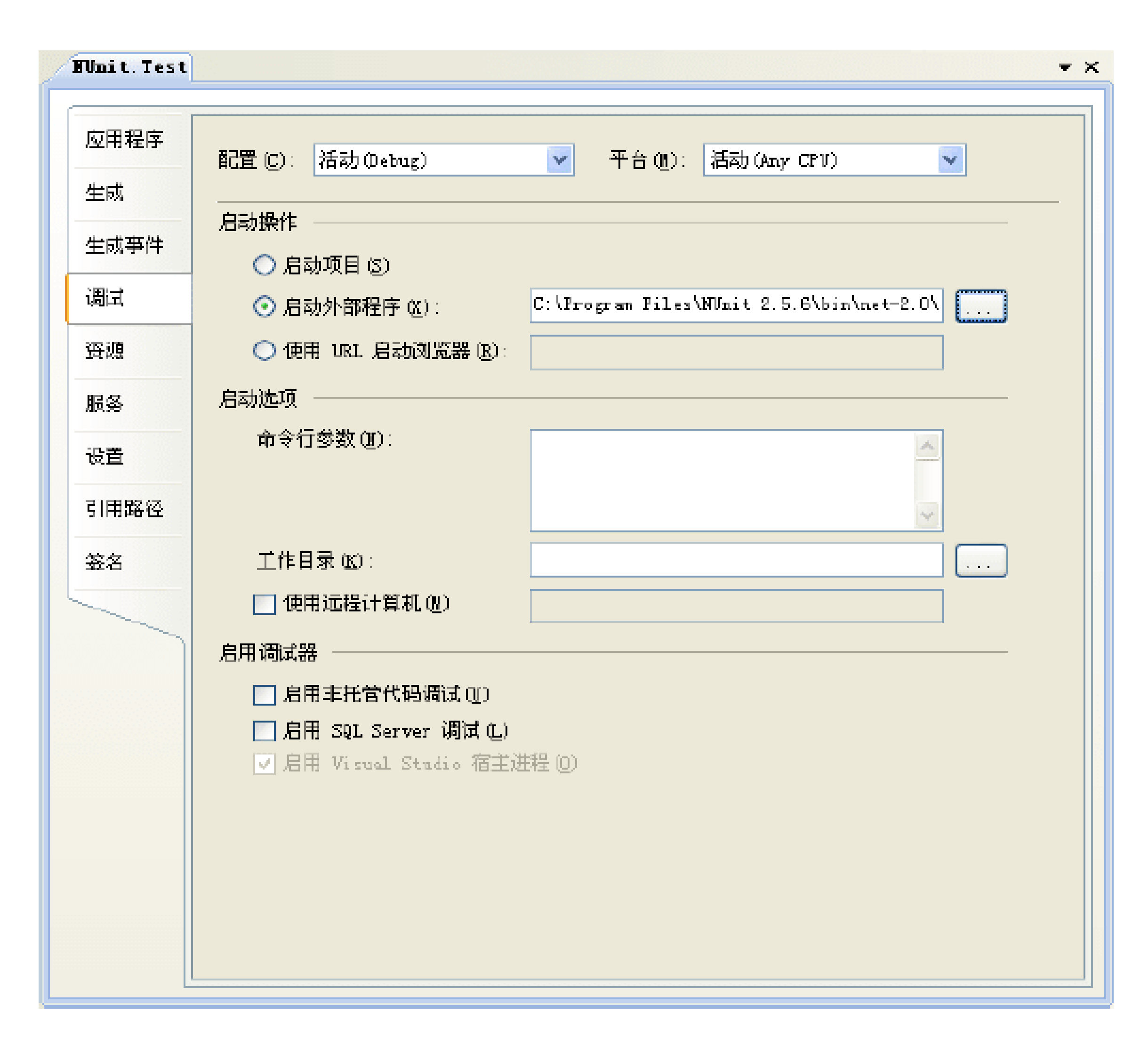


图 12-18　项目属性窗口

该操作是让该测试项目使用 NUnit 工具执行启动操作。

12.5.4　运行结果

在测试项目为默认启动项目的前提下(右击项目名称，选择“设为启动项目”)，单击工具栏中的“启动调试”按钮，系统会自动启动项目，并且打开 NUnit 工具，如图 12-19 所示。

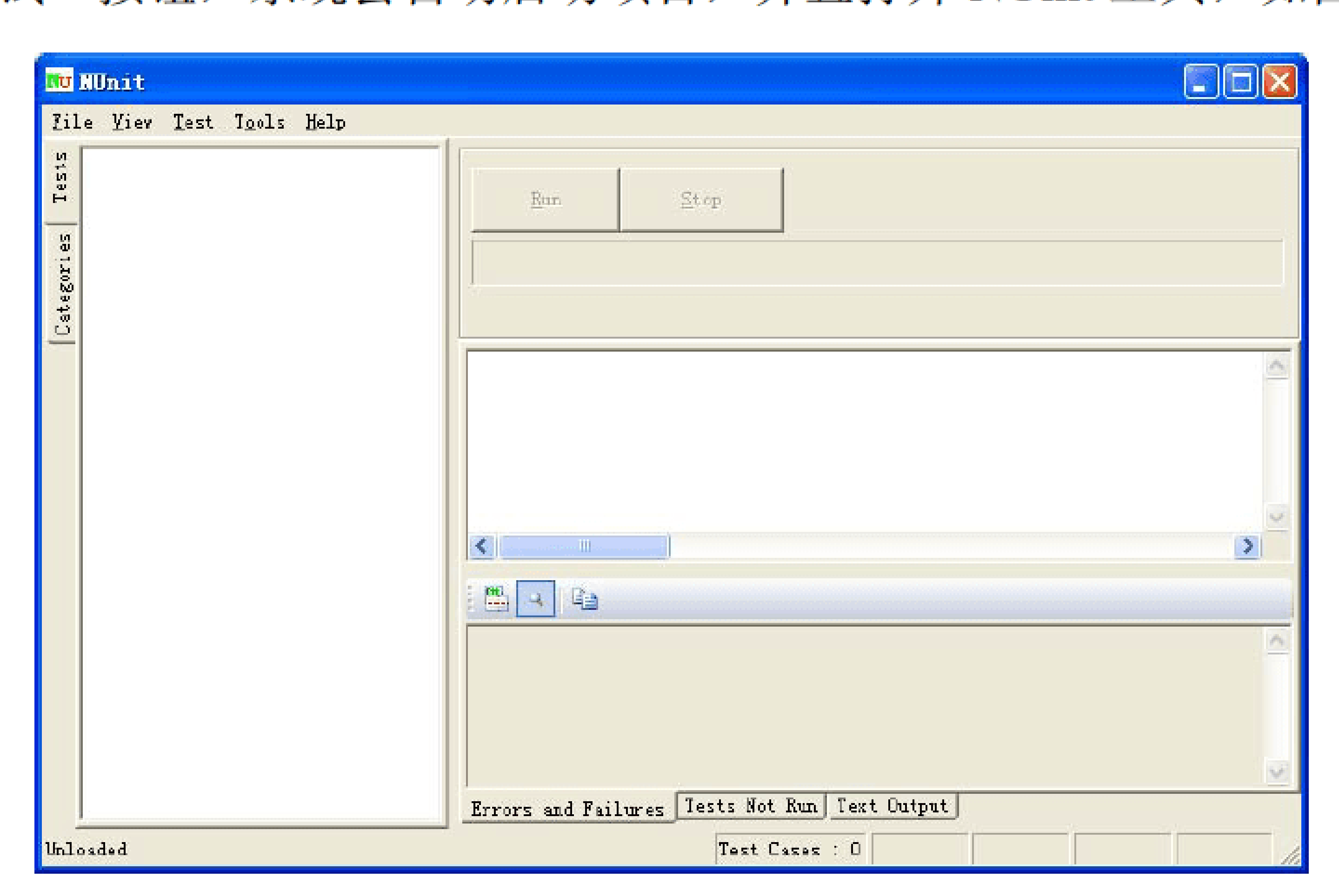


图 12-19　NUnit工具

第一次打开的时候没有加载项目，这里要打开我们的项目。

单击 File 菜单下的 Open Project 选项，打开 Open Project 对话框，找到我们项目中编译过的测试项目类库，如图 12-20 所示，单击“打开”按钮。

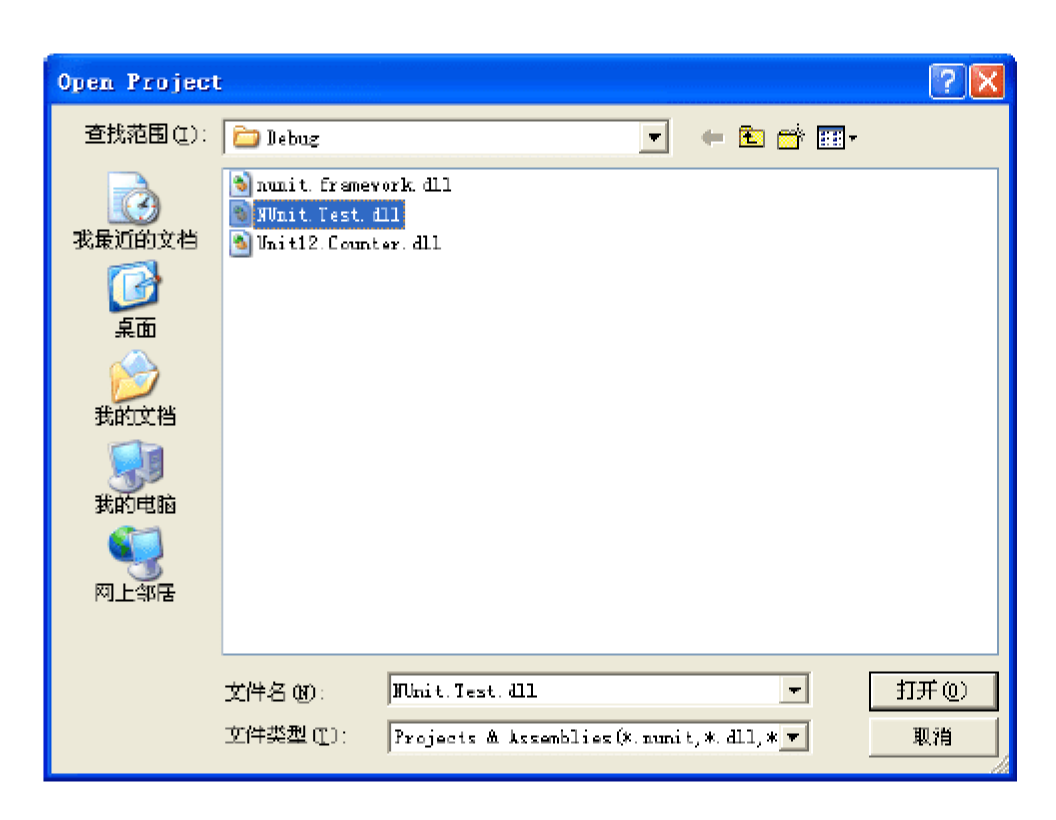


图 12-20　Open Project对话框

可以看到在 NUnit 主窗口左侧的树形列表框里按类库、命名空间、类、方法的顺序一级一级地展开我们的测试项目。

我们选中被测试的节点 TestCounter(类)，单击右侧的 Run 按钮开始执行测试，如图 12-21 所示。

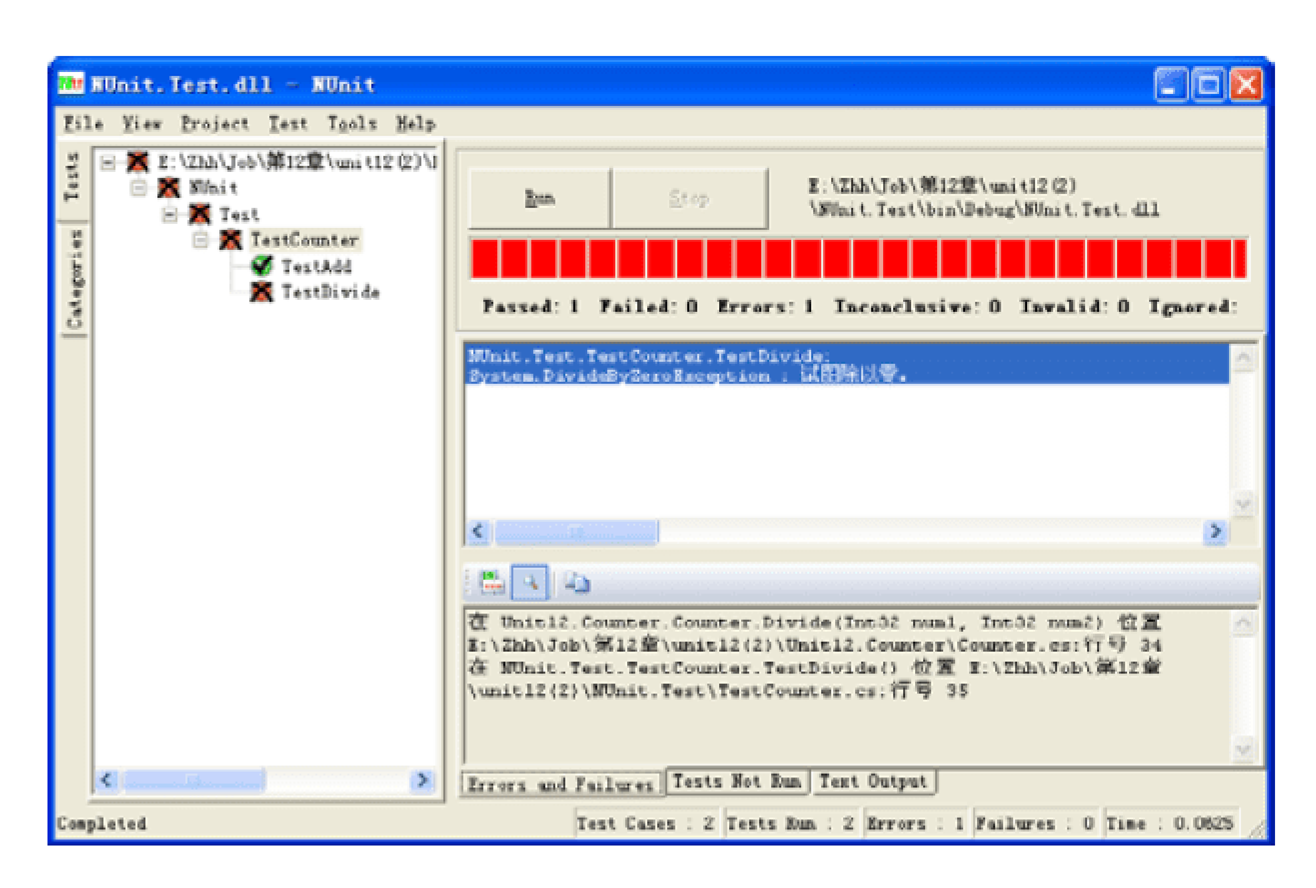


图 12-21　运行测试项目

这里发现 TestAdd 方法测试成功，而 TestDivide 方法抛出 System.DivideByZeroException 异常，警告被除数不能为 0。

这正合我们的意思，也说明这次测试完美地完成了。

12.5.5　实例分析


源码解析：

实例创建了一个测试类，对抽选出的计算器的业务方法进行测试。

在测试类中，我们使用 TestFixture 标记将一个普通的类标记为 NUnit 测试类，使用 SetUp 标记将测试类中的一个普通的方法标记为测试类的初始化方法，并使用 Test 标记将测试类中的一个普通的方法标记为测试方法。

在测试方法中，我们使用 Assert(断言)对测试结果进行验证。然后设置 NUnit 工具为项目的启动工具，进行单元测试。

12.6　ASP.NET的身份验证

身份验证，说白了就是谁有权限访问某些功能的问题。专业的说法是有特性访问需要的用户拿着系统颁发的验证凭据访问系统，系统根据凭据来验证用户权限的过程。

从 ASP.NET 2.0 开始，系统就提供了 4 种用户身份验证方式：Windows 验证、Forms 验证、Passport 验证和 None 验证，其中默认的是 Windows 验证。

在 web.config 文件中可以对身份验证方式进行设置。默认的配置节点如下：

```
<!--
    通过 <authentication> 节可以配置 ASP.NET
    用来识别进入用户的
    安全身份验证模式。
-->
<authentication mode="Windows" />
```

这里 mode 属性的值是 Windows，它还可以是 Forms、Passport 或 None。

不过要注意，在设置完验证方式后不能够动态修改。

1. Windows验证

Windows 验证一般用于企业内部网(Intranet)的站点中，该验证方式基于 Windows 系统账号和系统用户组，可以有效地以 Windows 系统的安全机制控制站点访问者对站点信息的访问。

企业可以为每一类有不同级别权限的员工分配不同的 Windows 用户账号，这些账号属于不同的用户组。如此便可对访问者的身份进行很方便的管理。

Windows 验证方式主要靠 Web 服务器工具 IIS 进行管理。IIS 验证用户身份之后，将请求传递给 ASP.NET。

在 ASP.NET 使用 WindowsAuthenticationModule 模块来实现 Windows 方式的验证，该模块根据相关信息构建 WindowsIdentity 对象。

同时 ASP.NET 从 IIS 传递过来的信息中构造 WindowsPrincipal 对象，将其附加到应用程序上下文中。程序利用创建的 WindowsIdentity 对象和 WindowsPrincipal 对象可以获取 Windows 验证过的用户的信息。

2. Forms验证

Forms 验证是大多数 Web 应用程序使用的用户身份验证方式，其主要原因就是灵活。

Forms 验证方式其实就是我们平时使用表单提交用户验证信息进行验证的方式。用户自定义用户界面，收集用户信息(比如常用的用户名和密码等)，最终还需要通过用户自定义的代码实现验证功能。

一般来说我们使用 Forms 验证完以后将用户标识信息保存到会话(Session)中，ASP.NET 的 Forms 验证主要是基于 Cookie 的。也就是说 Forms 验证将用户信息保存在 Cookie 中，然后随

响应发送到客户端，客户端再次请求后，解析用户的 Cookie，再进行验证。如果没有登录，当然就会验证失败，并跳转到登录页面。

3. Passport验证

.NET 的 Passport 是微软提供的一种服务。而 Passport 验证是由微软公司 1999 年公布的 Passport 服务发展而来的。

Passport 服务提供了一个在多个网站之间共享的验证机制，便于让用户使用一套凭据就可以进行访问和使用互联网上发布的 Web 站点和服务。

这就意味着，我们只要一次通过互联网上的 Passport 验证，就可以进行对其他具有权限的站点进行免验证访问。这样就可以不需要记住多个登录名和密码，而轻松地使用网络资源了。

Passport 验证就是基于 Passport 服务的一种验证方式。它依靠 Passport 中央服务器来验证用户，而且 Passport 中央服务器并不会授权或拒绝某用户访问某个启用 Passport 的站点，一切权限都由网站来控制。

微软公司为了推广 Passport 验证服务，还为开发人员提供了支持 Passport 的软件开发工具包，并且在.NET 中集成了类库。

4. None验证

顾名思义，None 在这里就是没有验证，就是不使用系统提供的验证机制对用户进行验证。那么，在使用这种验证方式的情况下，我们就只能自定义一些方法对用户的身份进行验证。

12.7 常见问题解答

12.7.1 ASP.NET加密有什么用



ASP.NET 加密有什么用？
网络课堂：http://bbs.itzcn.com/thread-9869-1-1.html

ASP.NET 加密有什么用呢？

把密码和用户名加密后存入数据库后是否可在数据库里复制出来然后登录？如果不能的话，能否把存入数据库的密码在 ASP.NET 里解密出来，然后再登录？

【解决办法】：ASP.NET 的加密用处很多，比如密码加密、密钥、加密文件、加密要在网络上传输的数据等。

你说的使用于密码加密的情况，一般可以用 MD5 等散列码的加密方式，这样“基本上”是不可逆转的。

这里之所以说是“基本上”，是因为现在还没有哪种加密技术是绝对地无法解密的，当年号称保密多少年的 MD5 也是在数天之内就被成功破解掉了。不过也不是说加密技术就形同虚设了，因为解密也是根据算法进行计算的，一般如果你的密码足够复杂、足够长，进行解密也不是一个简单的任务，所以说还算“比较”保密的。

至于第二个问题，完全没有必要，因为一般进行登录验证也是需要输入明文密码，加密后才进行比较的。

12.7.2 ASP.NET加密方法怎样解密



ASP.NET 加密方法怎样解密？

网络课堂：http://bbs.itzcn.com/thread-9871-1-1.html

见如下代码：

```
public static string jiami(string password)
{
    byte []buffer = System.Text.Encoding.UTF8.GetBytes(password);
    SHA1CryptoServiceProvider sha = new SHA1CryptoServiceProvider();
    byte []hash = sha.ComputeHash(buffer);
    StringBuilder passwordbullder = new StringBuilder(32);
    foreach (byte hashByte in hash)
    {
        passwordbullder.Append(hashByte.ToString("x2"));
    }
    return passwordbullder.ToString();
}
```

这是个加密方法，可以调用该方法加密文本框的值，怎样才能解密这个方法所加密的文本值呢？

【解决办法】：SHA1CryptoServiceProvider 是用于 Hash 的，前面 SHA1 就是一种 Hash 标准。

Hash 就是把一系列不定长二进制输出为一个定长的二进制，SHA1 的输出为 160 位。

Hash 用来做什么？现在一般是用来验证一段信息有没被篡改过或者就是保存密码的 Hash 值用于日后验证，又可防止别人查看这个记录窃取密码。Hash 值是不可逆的，不存在破解不破解的问题。

12.8 习 题

一、填空题

(1) 使用相同的密钥进行加密或解密的是__________。

(2) 使用加号对 10 个字符串进行连接，内存中将会产生__________个字符串对象。

(3) 现在我们需要对应用程序中的所有 PNG 格式的图片使用 MyHandler 命名空间下的 PNGHandler 进行处理，请补充 web.config 文件里的节点配置：

```
<add verb="*" path="*.png" type="__________" />
```

(4) .NET 中，对系统注册表操作时需要使用到__________类(写全路径)。

(5) 在 ASP.NET 管道中，可以有多个__________。

(6) 在 NUnit 中，使用__________(断言)对测试结果进行验证。

二、选择题

(1) 在对引用类型数据转换的时候，推荐使用__________关键字。

A. is B. as C. in D. are

(2) 下列加密算法中，不属于对称加密的算法是__________。

A. DES B. IDEA C. RC2 D. DSA

(3) 对于数据库连接对象的使用，有一个原则是__________。

A. 尽可能早的打开连接，尽可能晚的关闭连接

B. 尽可能晚的打开连接，尽可能早的关闭连接

C. 尽可能早的打开连接，尽可能早的关闭连接

D. 尽可能晚的打开连接，尽可能晚的关闭连接

(4) SQL 函数 row_number()是从 SQL Server__________版本才开始具有的函数。

A. 98 B. 2000 C. 2005 D. 2008

(5) 在 ASP.NET 中提供了几种数据验证方式，以下是 ASP.NET 提供的验证方式的有__________。(多选)

A. Windows B. Forms C. Passport D. None

(6) 在 NUnit 框架中，__________属性可以将一个类标记为包含测试功能的类。

A. TestFixture B. SetUp C. Test D. TestClass

(7) 在 web.config 中，修改 compilation 节点的__________属性可以设置和取消应用程序的调试模式。

A. test B. debug C. tiaoshi D. run

三、上机练习

上机练习：登录验证加密密码。

在一些用户管理系统中，用户密码信息是绝对不可以泄露的。所以，最好的设计方案是把密码进行加密后存储。这样即使数据库被非法入侵，也不会丢失用户的密码。也就防止了入侵者使用某个用户的密码正常登录后进行恶意操作。

所以这里我们实现一个登录加密密码的验证功能。数据库里存储已经使用 MD5 算法进行加密过的密码。

数据库用户账号表如图 12-22 所示。

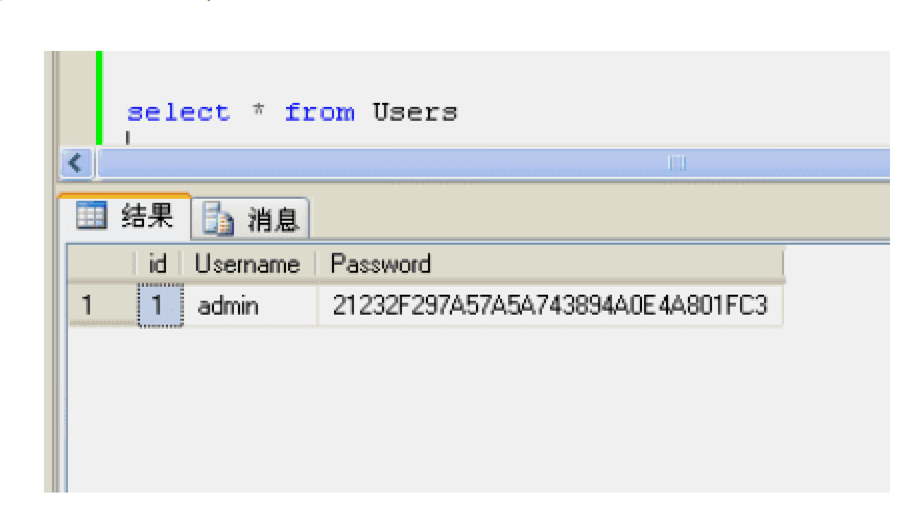


图 12-22 用户账号表

登录界面如图 12-23 所示。

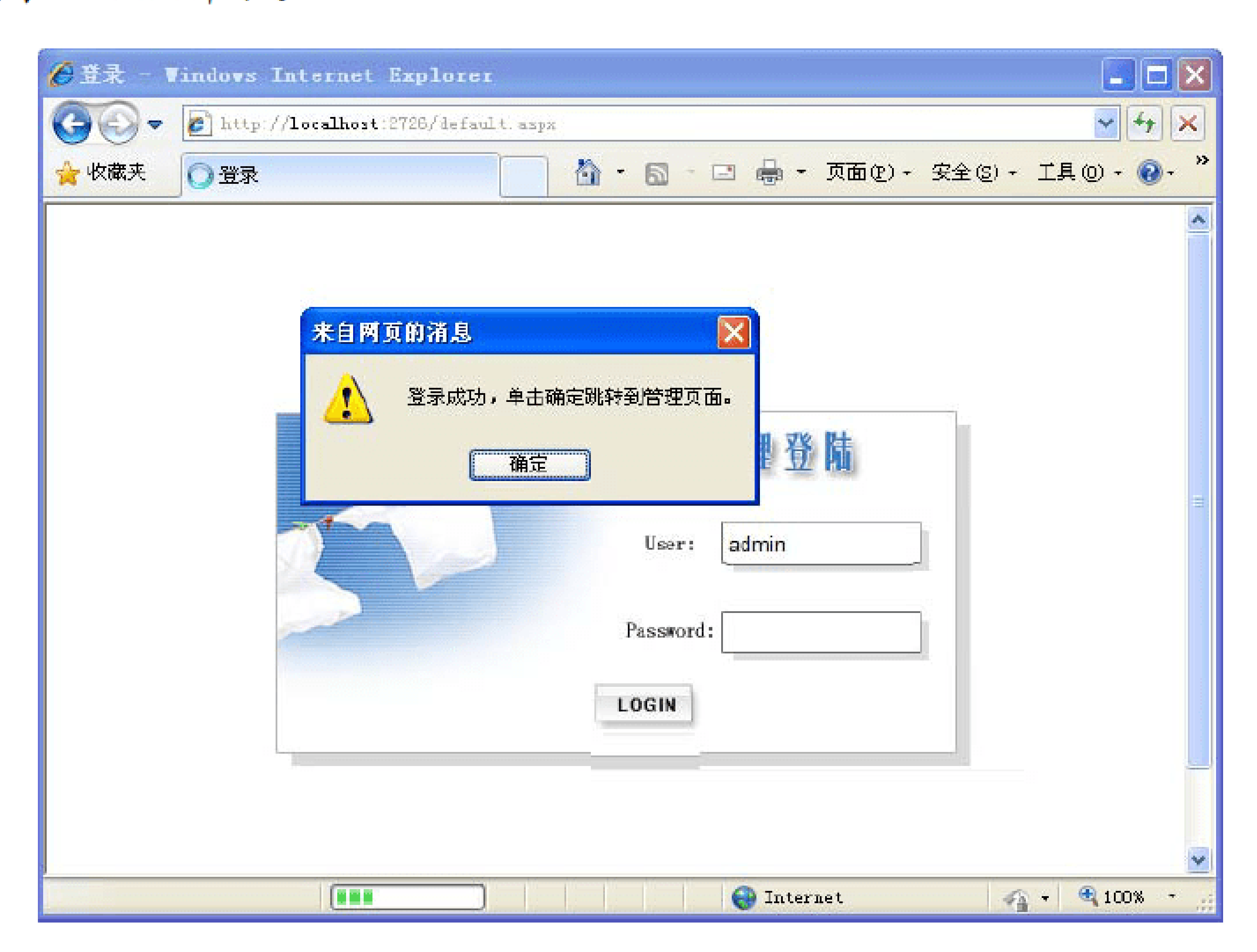


图 12-23　登录界面



第 13 章　ASP.NET还能干什么

内容摘要：

ASP.NET 的主战场是动态网站开发，无论是应对企业网站、门户网站、OA、或者是 CMS 系统，它都战无不胜。而且把 JSP 和 PHP 等这些竞争对手打得落花流水，比它们的优势不止多出两三点，简直有一箩筐……

为了扩大战果，ASP.NET 也在不断改善自己，寻求多元化发展，比如增加云的功能等。

在本章之前详细讨论了 ASP.NET 中制作动态网站时所需的各种技术，本章从实战项目中精选了几个案例来介绍 ASP.NET 的扩展功能。

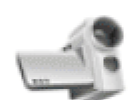
学习目标：

- 熟悉 ASP.NET 下向页面动态添加控件的方法
- 掌握外部 Web Service 的添加方法
- 熟悉如何引用和调用 Web Service
- 理解 Global.asax 文件的作用和代码编写
- 了解如何控制 Session 的生命周期
- 熟悉发送邮件的过程
- 掌握 AspNetPager 组件实现分页
- 掌握 PopupWin 组件如何处理弹出对话框
- 熟悉 log4net 组件记录日志的方法

13.1　动态生成页面控件

很多时候，开发人员设计程序时已经将页面所需要完成的功能制作好。剩下的工作，则是在运行时将预先设计的页面通过浏览器呈现给用户。也就是说，用户看到的页面布局不会发生变化。

有没有一种方案，能够在运行时由用户来决定如何显示，或者显示哪些东西呢？答案是肯定的，这就需要动态地向页面中添加控件，本节就讨论在 ASP.NET 下是如何完成这种功能的。

 **视频教学：**光盘/videos/13/Controls.avi　　　　**长度：11 分钟**

13.1.1　实例描述

在使用 ASP.NET 程序开发网站的过程中，大部分情况下需要在后台中用到上传功能。我们都知道，FileUpload 控件一次只能上传一个文件，然而在实际应用时，有时用户需要上传多个文件。这时一个一个地上传文件似乎有些麻烦，而且还不够人性化。如果把程序做成死的，即在页面上固定 3~5 个 FileUpload 倒也可以实现。但是这样很不灵活，同时也大大地影响了网页的美观。那么有没有一种既不影响页面美观，又十分灵活的解决办法呢？

方法还是有的，我们可以让 FileUpload 控件动态生成，用户想上传多少个文件，页面就动态生成多少个控件。

13.1.2　实例应用

【例 13-1】动态生成页面控件。

(1)　在页面上添加一个用于控制上传数量的 TextBox 组件，然后添加“批量上传”和“开始”上传两个按钮。

(2)　在页面的合适位置添加一个 ID 为 divControls 的容器，用于显示上传控件，添加另一个 ID 为 divMessage 的容器，用于显示上传结果信息。最终的布局效果如图 13-1 所示。



图 13-1　页面布局效果

(3) 页面添加了两个按钮，其中单击“批量上传”按钮后会动态生成若干个上传控件。该按钮的单击事件代码如下：

```
protected void btnCreate_Click(object sender, EventArgs e)
{       }
```

事件里面怎么没有代码呢？嗯，不用愁，代码在 Page_Load()事件里面呢。

(4) 由于控件是动态生成的，页面每一次 PostBack 时，都需要重新创建动态生成的控件。“批量上传”按钮只不过是引起了页面的一次 PostBack。Page_Load()事件的代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    btnOK.Enabled = false;
    if (this.IsPostBack )    //每次 PostBack 时，都需要重新创建 FileUpload 控件
    {
        int txtCount = int.Parse(Count.Text);
        if (txtCount >= 1 && txtCount <= 20) //限定最多可以动态生成 20 个上传控件
        {
            CreateFileUploadList(txtCount);
            Count.Enabled = false;
            btnCreate.Enabled = false;
            btnOK.Enabled = true;
        }
        else
        {
            Count.Enabled = true;
            btnCreate.Enabled = true;
            Count.Text = "1";
            Count.Focus();
        }
    }
}
```

(5) 在 Page_Load()事件中，程序调用了一个 CreateFileUpload()方法，该方法是动态生成控件的关键。代码如下：

```
private void CreateFileUploadList(int num)
{
    HtmlGenericControl div;
    HtmlGenericControl span;
    FileUpload fu;
    for (int i=0; i<num; i++)
    {
        div = new HtmlGenericControl("div");     //创建 div
        div.ID = "divfu" + i.ToString();
        div.Attributes["class"] = "item2";
        span = new HtmlGenericControl("span");      //创建 span
        span.ID = "spanfu" + i.ToString();
        span.InnerHtml = "第" + (i + 1).ToString() + "张照片:";
        fu = new FileUpload();                     //创建 FileUpload
```

```
        fu.ID = "fu" + i.ToString();
        fu.CssClass = "fileup";
        div.Controls.Add(span);                    //添加控件到容器
        div.Controls.Add(fu);
        divControls.Controls.Add(div);
    }
}
```

(6) 上面的代码实现了动态生成上传控件的功能，下面该实现上传功能了。接下来，我们在第二个按钮的单击事件中添加如下代码：

```
protected void btnOK_Click(object sender, EventArgs e)
{
    FileUpload fu;
    StringBuilder sbResult = new StringBuilder();
    int txtCount = int.Parse(Count.Text);
    for (int i=0; i<txtCount; i++)
    {
        //注意：这里必须通过上层容器来获取动态创建的 FileUpLoad
        fu = divControls.FindControl("fu" + i.ToString()) as FileUpload;
        UploadPicFile(fu);     //调用上传图片方法
        sbResult.AppendFormat("第{0}张图片上传状态：{1}.<br />", i + 1, msg);
    }
    //在页面的 divMessage 标签中显示每张图片的上传信息
    divMessage.InnerHtml = sbResult.ToString();
}
```

(7) 上述代码中调用了一个用于上传图片的方法 UploadPicFile()。关于图片上传的实例，我们在前面的章节中多次用到过，这里不再给出代码，具体代码请参照本节的源代码。

13.1.3 运行结果

下面，我们运行一下页面，看看效果如何。如图 13-2 所示为我们在文本框输入“6”并单击“批量上传”按钮后，页面动态生成上传控件的效果。



图 13-2 页面动态生成 6 个FileUpload控件

随后，我们单击“浏览”按钮，开始选择要上传的图片。最后单击“开始上传”按钮，页面显示每张图片的上传信息，效果如图 13-3 所示。

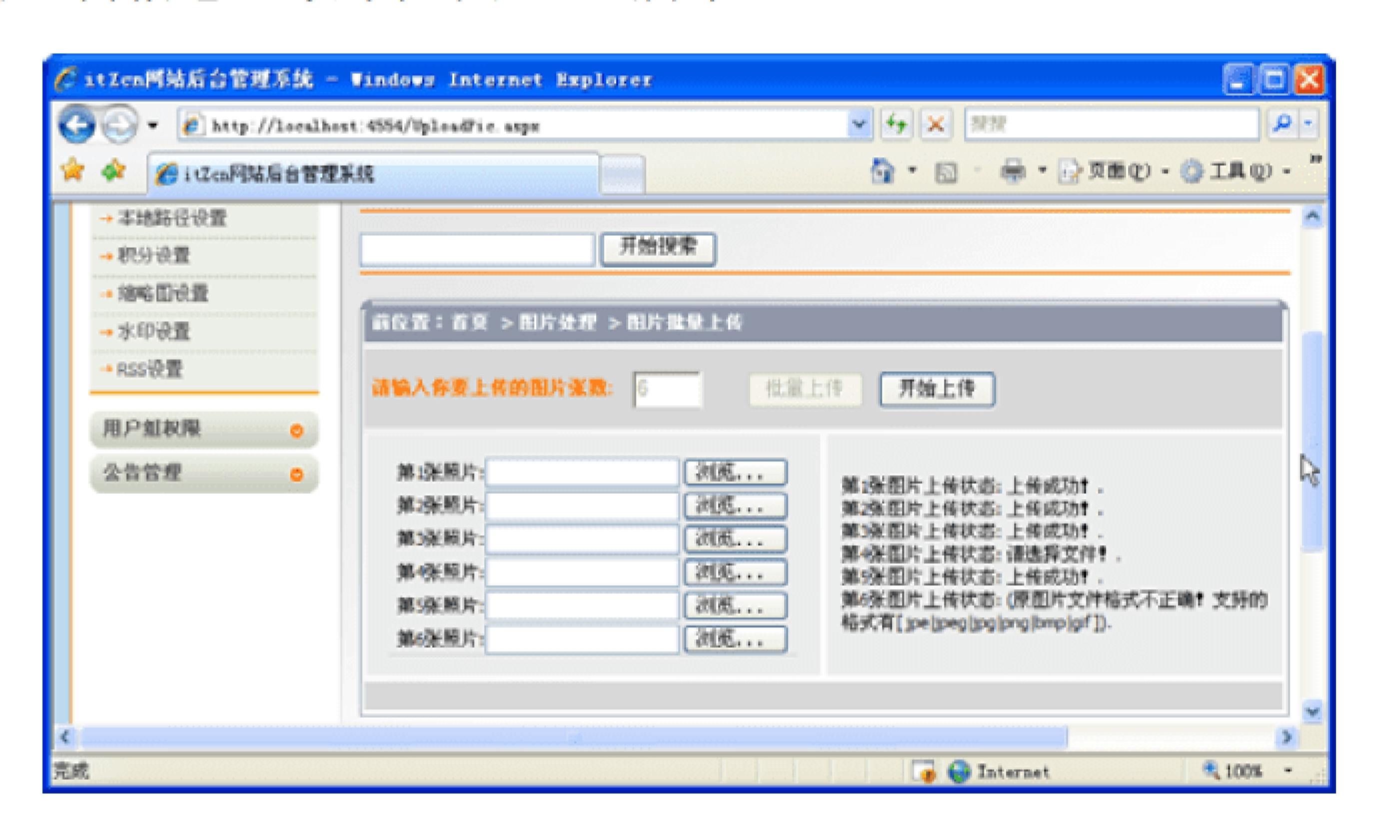


图 13-3　单击“开始上传”按钮后页面的效果

13.1.4　实例分析



源码解析：

在本实例中，我们主要使用了 Controls.Add()方法来向页面动态添加上传控件。该方法比较简单，关键在于如何给动态生成的控件添加属性和值，以及如何控制它的样式。

实例中我们声明了两个标签，一个控件，然后通过变量分别给它们的 ID 赋值，最后通过 Controls.Add()依次添加到容器中。

最终生成的 HTML 代码类似下面这样：

```
<div id="divfu0" class="item2">
   <span id="spanfu0">第 1 张照片:</span>
   <input type="file" name="fu0" id="fu0" class="fileup" />
</div>
```

13.2　ASP.NET调用Web Service

Web Service，顾名思义就是 Web 服务，它的好处就是可以实现远程调用、数据共享等。由于它部署在独立的服务器上，所有用户都通过接口访问而看不到具体怎么实现，这对技术保护和安全以及维护带来很大的好处。

本节我们将来看看 ASP.NET 如何调用一个在别人服务器上的 Web Service。



视频教学： 光盘/videos/13/useWebService.avi　　　　长度：6 分钟

13.2.1　基础知识——如何调用Web Service

在 VS2008 中，调用 Web Service 是非常容易的，我们只需要几步简单的操作，便可完成对 Web Service 的调用。具体步骤如下。

(1) 在 VS2008 中，新建一个 ASP.NET Web 应用程序。

(2) 在“解决方案资源管理器”窗格中右击项目名称，选择“添加 Web 引用”项，弹出“添加 Web 引用”对话框，如图 13-4 所示。

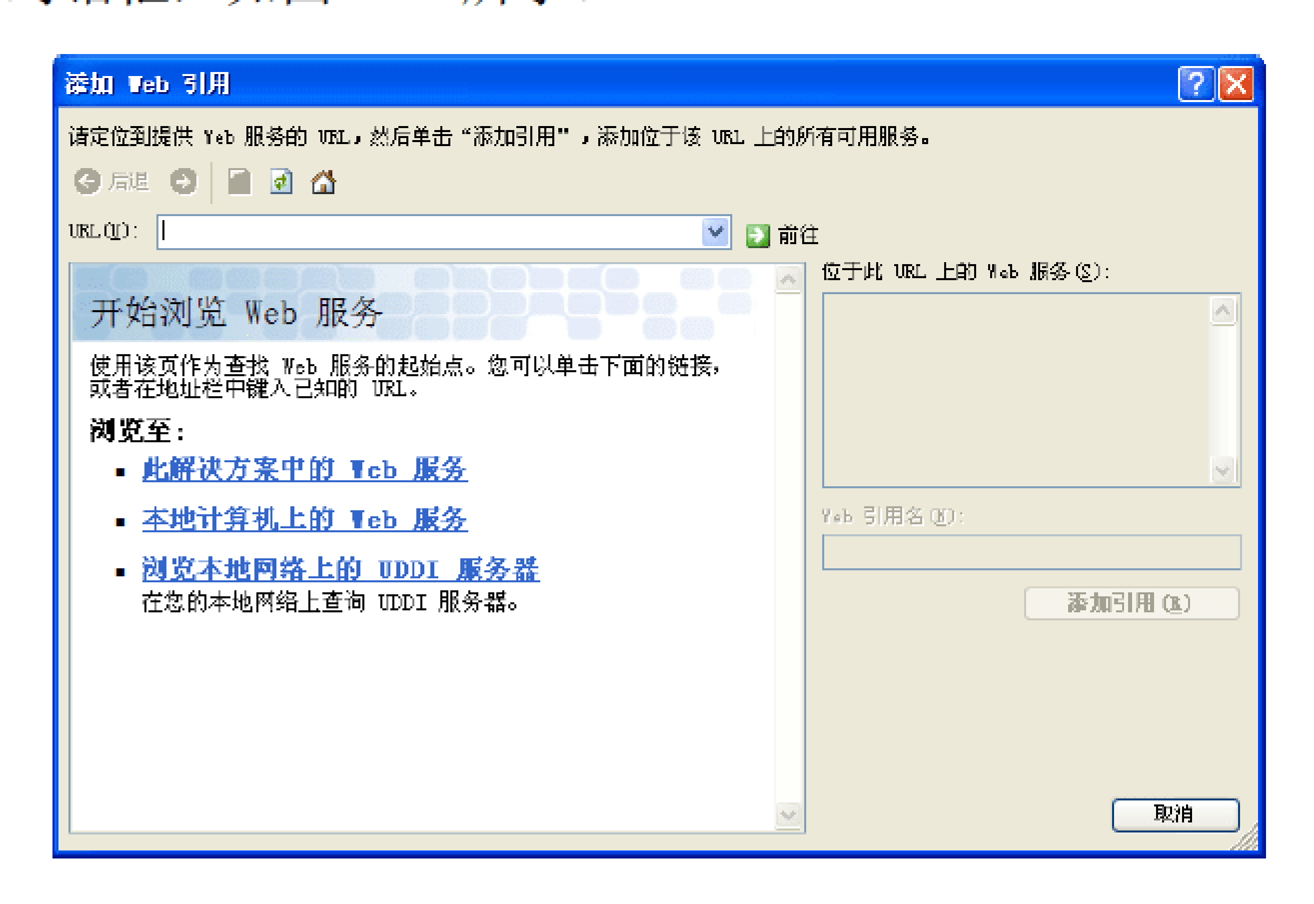


图 13-4　“添加Web引用”对话框

(3) 在 URL 地址栏中输入一个能够提供 Web Service 的 URL 地址(该服务可以是本地计算机上的 Web 服务，也可以是远程网络上的 Web 服务)。

在这里我们输入“http://www.webxml.com.cn”，单击“前往”按钮后，对话框中会显示如图 13-5 所示的信息。

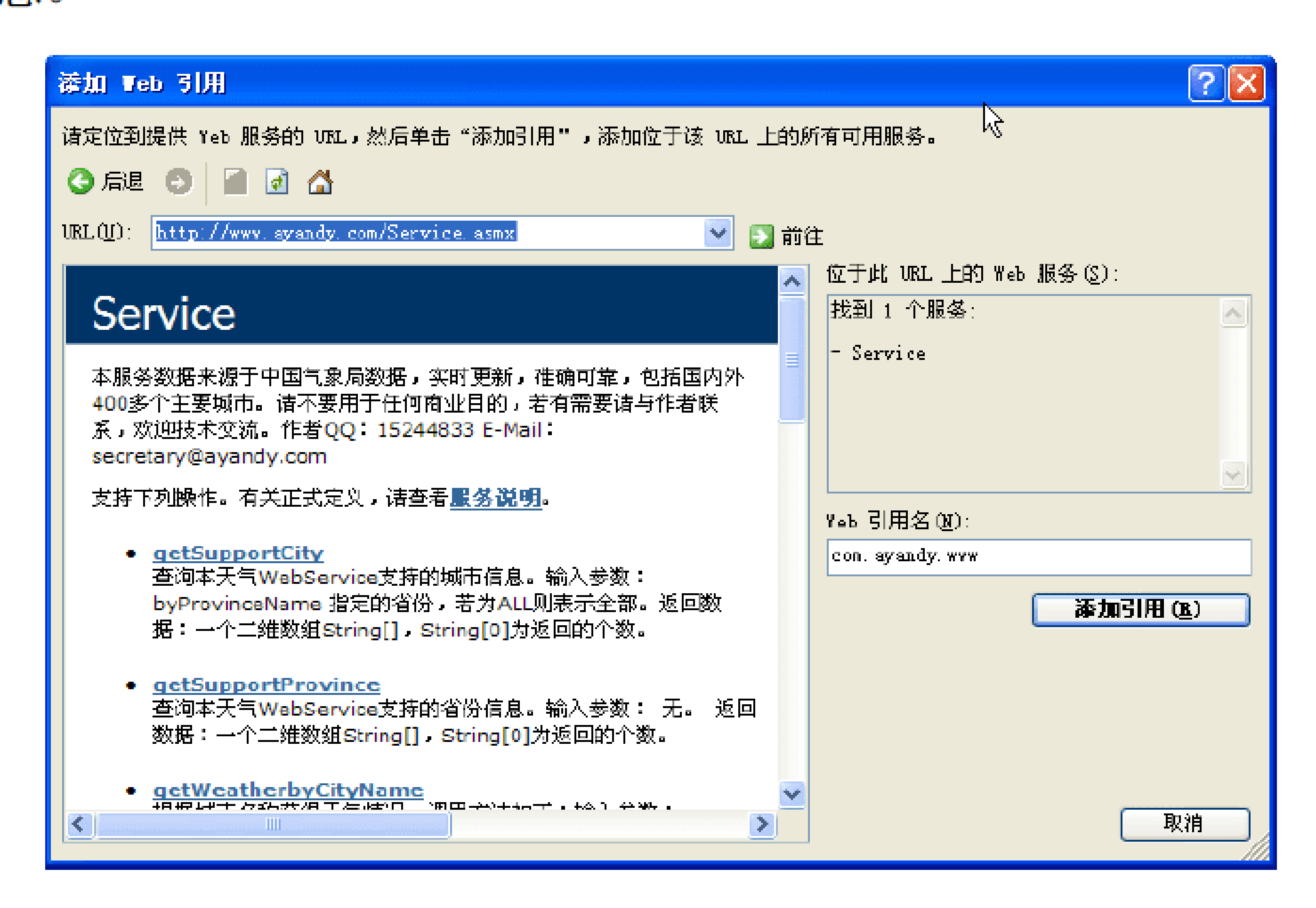


图 13-5　设置Web引用名称

(4) 然后，在“Web 引用名”文本框中为该引用定义一个名称(也可以保持默认值)，我们将在代码中使用该名称以访问所选择的 Web 服务。

(5)　最后单击“添加引用”按钮，此时项目中会自动添加一个 Web References 目录，在该目录下会自动生成一个代理类——Reference.cs。

(6)　在编程过程中使用该类和使用本地普通类没有什么区别，如下代码便实例化了一个代理类，并调用该类的一个方法：

```
// Obj.WeatherWebService 为一个 Web Service 的引用名称
Obj.WeatherWebService Weather = new Obj.WeatherWebService();   //obj
ds = Weather.getSupportDataSet();
```

至此，ASP.NET 调用 Web Service 的过程就完成了。

13.2.2　实例描述

上面我们讲到了如何调用 Web Service，那么调用 Web Service 能干什么，以及调用它有什么好处呢？我想这可能是大家心中最大的疑问。

关于这两个问题，在网上随便一搜都能找到很多详尽的解答，这里不做过多回答。

下面，让我们通过实例——调用 Web Service 实现查询各地天气预报，来更好地理解一下这两个问题。

13.2.3　实例应用

【例 13-2】调用 Web Service 实现查询各地天气预报。

(1)　新建一个命名为“MyWeather”的 Web 项目。

(2)　为该项目添加一个 Web Service 引用，URL 地址为“http://www.webxml.com.cn”，引用名称改为“Obj”。

(3)　调用 Web Service 完成之后，就该布局页面了。在页面设计窗口制作如图 13-6 所示的布局，用于显示天气信息。

由于页面布局比较简单，代码不再具体给出，请参考本节的源代码。

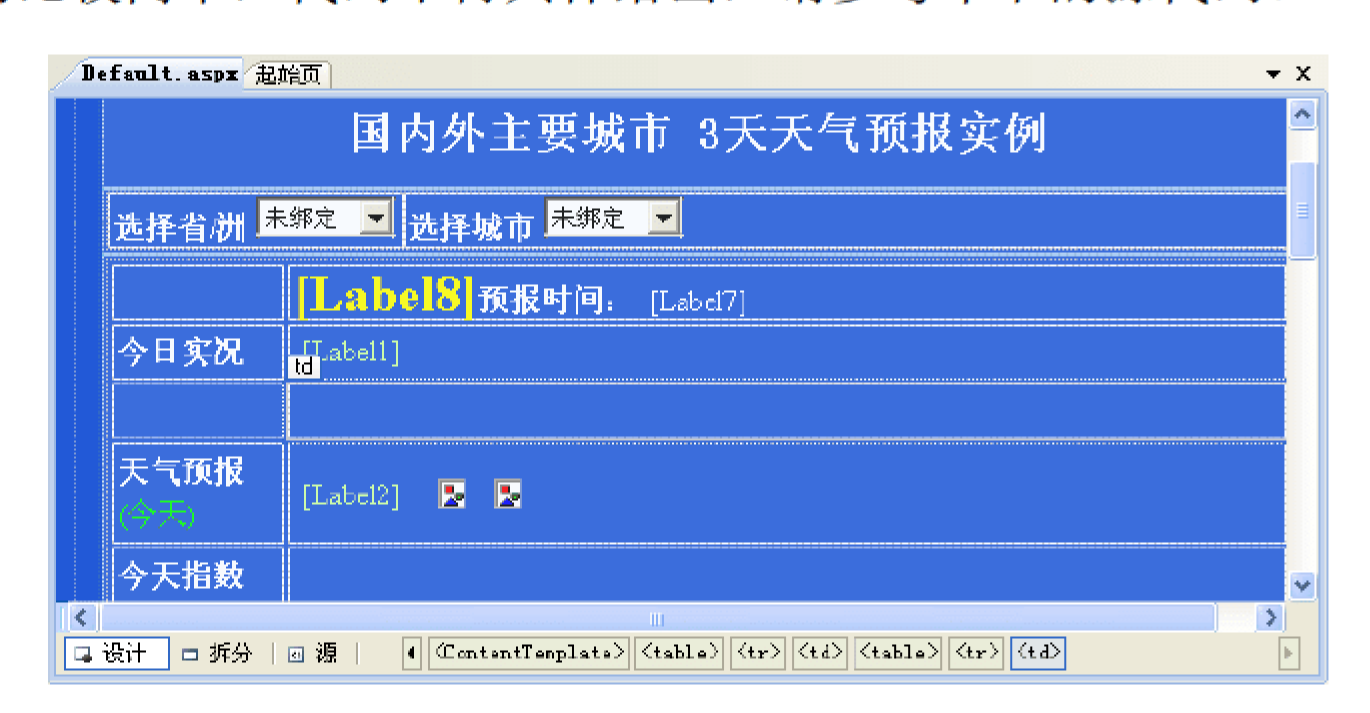


图 13-6　页面布局

(4)　接下来，新建一个公共类，命名为“WeatherClass”。该类通过使用代理类“Obj”取得城市信息以及对应城市的天气信息，并将它们返回。

该类的具体代码如下：

```
public class WeatherClass
{
    // 取得支持的城市，若高速缓存中不存在，则直接调用服务读取
    public static DataSet getSupportDataSet()
    {
        DataSet ds = (DataSet)HttpContext
          .Current.Cache["dataSetCache"]; //从高速缓存中读取记录集
        if (ds == null) //若无此缓存
        {
           Obj.WeatherWebService Weather = new Obj.WeatherWebService();
           ds = Weather.getSupportDataSet();
           HttpContext.Current.Cache.Insert(
            "dataSetCache", ds, null, DateTime.Now.AddMinutes(30),
            TimeSpan.Zero,
            System.Web.Caching.CacheItemPriority.High, null); //设置缓存
        }
        return ds;
    }
    // 根据城市代码取得城市天气，有高速缓存
    public static string[] GetCityWeather(string cityCode)
    {
        string cacheName = "Weather" + cityCode.Trim();
        string []WeatherArray =
          (string[])HttpContext.Current.Cache[cacheName];
        if (WeatherArray == null)
        {
            Obj.WeatherWebService weather = new Obj.WeatherWebService();
            WeatherArray = weather.getWeatherbyCityName(cityCode.Trim());
            HttpContext.Current.Cache.Insert(cacheName, WeatherArray, null,
              DateTime.Now.AddMinutes(30), TimeSpan.Zero,
              System.Web.Caching.CacheItemPriority.High, null);
        }
        return WeatherArray;
    }
}
```

(5) WeatherClass 类设计完毕。下面进入到网页的后台代码设计窗口，在 Page_Load()事件中添加如下代码来初始化网页：

```
protected void Page_Load(object sender, EventArgs e)
{
    try
    {
        DataSet mds = WeatherClass.getSupportDataSet();
        if (!Page.IsPostBack)
        {
            DataTable dt = mds.Tables[0];             //支持的省洲
            this.Province.DataSource = dt;
```

```
            this.Province.DataValueField = "ID";      //Value 存储省洲代码
            this.Province.DataTextField = "Zone";     //显示省洲名称
            this.Province.DataBind();
            City.SelectedIndex = 1;
            CityDataBind("1");                    //默认直辖市
            GetWeatherByCode("54511");            //默认北京
        }
    }
    catch (Exception)
    {
        Title.Text = "发现一个错误";
    }
}
```

(6) 在初始化网页的时候，上段代码调用了一个 GetWeatherByCode()方法，该方法用于根据城市代码取得该城市的天气信息，并绑定天气信息到相关显示控件。该方法的具体代码如下：

```
protected void GetWeatherByCode(string cityCode)
{
    String []wa =
      WeatherClass.GetCityWeather(cityCode.Trim()); //天气的字符串数组
    Label1.Text = wa[10];
    Label2.Text =
      wa[6] + "   " + wa[5] + "   " + wa[7];
    Label3.Text =
      wa[13] + "   " + wa[12] + "   " + wa[14];
    Label4.Text =
      wa[18] + "   " + wa[17] + "   " + wa[19];
    Label5.Text = wa[11].Replace("\n", "<br />");
    Label6.Text = wa[22].Replace("\n", "<br />");
    Label7.Text =
      DateTime.Parse(wa[4]).ToString("yyyy 年 MM 月 dd 日 dddd HH:mm");
    Label8.Text = wa[0] + " / " + wa[1];
    Image1.ImageUrl = "~/images/weather/" + wa[8];
    Image2.ImageUrl = "~/images/weather/" + wa[9];
    Image3.ImageUrl = "~/images/weather/" + wa[15];
    Image4.ImageUrl = "~/images/weather/" + wa[16];
    Image5.ImageUrl = "~/images/weather/" + wa[20];
    Image6.ImageUrl = "~/images/weather/" + wa[21];
    CityPhoto.ImageUrl = "http://www.cma.gov.cn/tqyb/img/city/" + wa[3];
    CityPhoto.AlternateText = City.SelectedItem.Text;
}
```

(7) 在网页中，下拉列表省份名称和城市名称是级联显示的。也就是说当我们选择了省份以后，对应城市的名称才会出现到城市名称的下拉列表中。实现该功能的代码如下：

```
// 省洲选择变化的事件处理
protected void Province_SelectedIndexChanged(object sender, EventArgs e)
{
    CityDataBind(Province.SelectedItem.Value.Trim());
```

```
}
// 绑定城市的数据，根据省洲代码对显示进行筛选
protected void CityDataBind(string zoneID)
{
    DataView dv = new DataView(WeatherClass.getSupportDataSet().Tables[1]);
    dv.RowFilter = "[ZoneID] = " + zoneID; //筛选的条件是 ZoneID=省洲代码
    City.DataSource = dv;
    City.DataTextField = "Area";
    City.DataValueField = "AreaCode";
    City.DataBind();
    City.Items.Insert(0, new ListItem("选择城市", "0"));
    City.SelectedIndex = 0;
}
```

(8) 当我们选择了一个城市后，对应城市的天气信息便会显示在网页中。这个功能是如何实现的呢，请参照如下代码：

```
// 城市选择变化的事件处理
protected void City_SelectedIndexChanged(object sender, EventArgs e)
{
    if (City.Items[0].Value == "0")
    {
        City.Items.RemoveAt(0);
    }
    GetWeatherByCode(City.SelectedItem.Value.Trim());
}
```

13.2.4 运行结果

至此，整个项目设计完毕，下面我们运行一下网页，看看效果如何。在网页中，我们查询一下河南省郑州市的天气，页面效果如图 13-7 所示。



图 13-7 天气查询的效果

13.2.5　实例分析



源码解析：

经过这个实例的练习，读者对调用 Web Service 的步骤以及使用 Web Service 的好处有了进一步的认识。

在实例中，我们并没有在下拉列表中指定省名称和城市名称；我们更不可能把每个城市的天气信息都保存在本地。然而通过调用 Web Service ，这些看似不可能的功能通过简单的几行代码便实现了。

另外，读者在调用 Web Service 的过程中，一定要注意 Web Service 对应提供的方法名称、参数类型和个数，以及返回值类型等细节。

13.3　防止用户多次登录的方法

在 Web 开发时，有的系统要求同一个用户在同一时间只能登录一次，也就是如果一个用户已经登录了，在退出之前如果再次登录的话需要报错。本节将介绍最常见的处理方法之一，即利用 Session 的实现方案。



视频教学：光盘/videos/13/noReLogin.avi　　　　长度：8 分钟

13.3.1　实例描述

前几年有这样一个网站：上面有好多有价值的资源，但是只有会员才能下载。于是想下载的游客纷纷开始注册成为会员。会员越多，人气越旺，于是站长大赚了一把广告费。可是没过多久，站长发现新注册的会员越来越少，而资源还是被疯狂地下载。站长苦思冥想，想不出所以然来。最后经一高人指点，原来该站的会员可以重复登录……

上面的例子告诉我们，在网站中“用户可以重复登录”确实是个不小的漏洞。不过没关系，防止的方法也有很多。本实例中，我们的思路是，当有用户重复登录时，要禁止他再次登录并提示该账号已经登录。

13.3.2　实例应用

【例 13-3】防止用户多次登录。

(1)　首先，制作一个登录页面“Login.aspx”，在该页面中分别设置用户名和密码以及按钮。这一部分代码比较简单，不再具体给出，可以参考本节的源代码。

(2)　接下来，在“登录”按钮的单击事件中加入如下代码，判断输入的信息是否正确：

```
protected void loginBtn_Click(object sender, ImageClickEventArgs e)
```

```
{
    string name = this.username.Text.Trim();
    string pwd = this.userpwd.Text.Trim ();
    //从数据库中查询用户信息
    string con =
      ConfigurationManager.ConnectionStrings["con"].ConnectionString;
    string sqlStr =
      string.Format("select count(*) from UserTable where username='{0}' and
       userpwd='{1}'", name, pwd);
    SqlConnection conn = new SqlConnection(con);
    conn.Open();
    SqlCommand cmd = new SqlCommand(sqlStr, conn);
    int n = Convert.ToInt32(cmd.ExecuteScalar()); //返回结果集中的第一行第一列
    if (n > 0)       //验证是否登录成功
    {
         //登录成功
    }
    else  //登录失败
    {
        ClientScript.RegisterStartupScript(
          this.GetType(), "", "<script>alert('用户名或密码错误！')</script>");
    }
}
```

(3) 登录成功之后，还需要判断是否存在在线用户列表，代码如下：

```
ArrayList list;
list = Application.Get("User_list") as ArrayList;
if (list == null)   //判断在线用户列表是否为空
{
    //为空时，则创建，再将当前用户添加到此列表
}
else
{
    //不为空时，判断当前用户是否在列表中。存在时提示，否则添加
}
```

(4) 这一步编写用户列表为空时的代码，此时应该创建一个列表：

```
list = new ArrayList();          //创建列表
```

(5) 如果有用户已经登录，此时列表肯定不为空。判断时则需要循环用户列表中的每一个名字，再与当前名称进行比较。如果一样则说明已经登录，弹出提示对话框。否则在最后将当前用户添加到用户列表中，这部分代码如下所示：

```
for (int i=0; i<list.Count; i++)        //循环当前用户列表
{
    if (name == list[i].ToString())      //判断当前用户是否在登录列表中
    {
        ClientScript.RegisterStartupScript(this.GetType(), "",
          "<script>alert('该用户:" + name + "已登录！')</script>");
```

```
        list.Remove(name);
        Application.Add("User_list", list);
    }
}
}
Session["name"] = name;         //获取用户名
list.Add(name);                 //将用户名添加到列表
Application.Add("User_list", list);         //更新列表
//转到 index.aspx 页面
ClientScript.RegisterStartupScript(this.GetType(), "",
  "<script>location.href = 'index.aspx'</script>");
```

(6) 在上述代码中，如果用户信息验证成功，页面将跳转到 index.aspx 主页面。在该页面的适当位置插入一个标签，用于显示当前用户名。添加一个按钮，用于注销用户。前台的布局比较简单，下面看一下后台的相关代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    msg .Text ="当前用户名: "+Session ["name"];       //显示当前用户
}
protected void closeBtn_Click(object sender, EventArgs e)
{
    Session.Abandon();                                  //安全退出时清空 session
    Response.Write("<script> parent.location.href='/Login.aspx';</script>");
    //转向登录页
}
```

(7) 下面为该项目添加一个全局应用程序文件“Global.asax”，在 Session_End()事件中添加如下代码：

```
protected void Session_End(object sender, EventArgs e)
{
    string name = (String)Session["name"];              //获取当前用户名
    ArrayList list = (ArrayList)Application.Get("user_list");  //获取用户列表
    if (name!=null && list!=null)
    {
        list.Remove(name);                          //将用户名从列表中删除
        Application.Add("user_list", list);
    }
}
```

13.3.3　运行结果

经过以上几步的操作，一个简单的防止用户重复登录的网站便完成了。下面运行登录界面，验证一下功能是否实现了。在登录界面，我们以用户名“admin”登录到系统主页，效果如图 13-8 所示。



图 13-8　用户登录成功

接下来，用浏览器再打开一个登录界面，仍然以用户名“admin”进行登录。此时页面将弹出提示用户已经登录的信息，说明防止用户重复登录功能已经实现，效果如图 13-9 所示。



图 13-9　重复登录提示

13.3.4　实例分析



源码解析：

本实例防止用户重复登录的思路如下：

当用户登录成功后，将用户名存放到 ArrayList 类型的 Application[“User_list”]里面；当用户再次登录时，查看动态数组中是否有对应的用户名，如果没有则可以登录，如果有则弹出已登录提示。

当用户注销时，调用 Session.Abandon()方法，并在 Global.asax 里的 Session_End 事件中将用户名从动态数组中删除。

13.4　构建电子邮件发送系统

发送电子邮件是邮箱系统中最基本的功能。现在，它同时也越来越作为网站的通用功能，在用户注册后发送一封邮件到注册时填写的电子邮件地址。

通常验证电子邮件真实有效的办法是：当用户填写的注册资料经过网站初步格式验证之后，用户并不能利用此账号登录。系统会向用户注册时填写的电子邮件地址发送一封电子邮件，邮件中给出一个链接。只有当用户单击了这个链接之后才能登录到网站，如果用户填写的电子邮件地址不是真实有效的或者不是他本人的，就不会收到这封电子邮件。这样仍然不能登录，这一步一般称之为电子邮件激活。



视频教学：光盘/videos/13/MailMessage.avi　　　　长度：8 分钟

13.4.1　基础知识——MailMessage类

MailMessage 类位于 System.Net.Mail 命名空间下，结合同命名空间下的 SmtpClient 类可以实现邮件的发送的功能。

使用 MailMessage 类实例提供的属性可以设置电子邮件的发件人、收件人、主题、内容以及是否有附件等。

这些属性如下所示。

- From：发件人
- To：收件人
- CC：抄送人
- Bcc：密件抄送人
- Attachments：附件
- Subject：邮件主题
- Body：邮件正文

经过上述步骤设置一个完整的电子邮件后，将它作为 SmtpClient 类 Send()方法的参数完成发送过程。

技术文档	SMTP的含义
SMTP 的全称是 Simple Mail Transfer Protocol，即简单邮件传输协议，它是一组用于从源地址到目的地址传输邮件的规范，通过它来控制邮件的中转方式。 SMTP 协议属于 TCP/IP 协议簇，它帮助每台计算机在发送或中转信件时找到下一个目的地。 SMTP 服务器就是遵循 SMTP 协议的发送邮件服务器。	

13.4.2 实例描述

如果要用 ASP 来做一个 E-mail 发送系统，首先想到的一定是借助第三方开发的组件，但这样做会耗费你很多的银子。

当然也可以用 Windows 内置的 CDONTS.NewMail 对象来传送 E-mail。它虽然是免费的，但却十分依赖操作平台，CDONTS.NewMail 对象只存在于 Windows 2000/NT 下，在 Windows 95/98 下没有这个对象。

自从 Microsoft 公司推出了 ASP.NET，E-mail 的发送已经成为 Web 程序设计的基本对象。这里就来讨论如何做一个功能简单的 E-mail 发送系统。

13.4.3 实例应用

【例 13-4】构建电子邮件发送系统。

(1) 首先新建一个名为 MailHelper 的实体类，在该类中封装对邮件的基本操作。

(2) 在类中添加与邮件相关的属性，包括附件地址、附件数组、信件内容、发件人、收件人、主题、SMTP 服务器、登录用户名和密码等。

(3) 继续补充 MailHelper 类，添加私有方法 JudgeAtt()判断是否存在附件，返回 bool 类型。

(4) 添加公共方法 JudgeConnect()判断是否可以连接上网络，返回 bool 类型。



JudgeAtt()和 JudgeConnect()这两个方法的具体代码这里不再给出，可以参考实例源文件。

(5) 添加无返回值的公共方法 SendMail()，用于实现发送邮件功能：

```
public void SendMail()
{
    try
    {
        // 邮件内容设置
        MailAddress MFrom = new MailAddress(this.Form);
        MailAddress MTo = new MailAddress(this.TO);
        MailMessage MailM = new MailMessage(this.Form, this.TO);
        MailM.Subject = this.Subject;
        MailM.Body = this.Body;
        // 添加附件
        if (Att != null)
        {
            if (this.JudgeAtt(Att.Trim()) == true)
            {
                if (Att.Trim().Length > 0)
                {
                    Attachment At = new Attachment(this.Att);     // 附件
                    MailM.Attachments.Add(At);
                }
```

```
            }
        }
        if (AttList != null)
        {
            if (AttList.Length > 0)
            {
                for (int i=0; i<AttList.Length; i++)
                {
                    if (this.JudgeAtt(AttList[i]) == true)
                    {
                        Attachment At1 = new Attachment(AttList[i]);
                        MailM.Attachments.Add(At1);
                    }
                }
            }
        }
        MailM.IsBodyHtml = true;     // 设置邮件支持 HTML
        MailM.BodyEncoding = System.Text.Encoding.GetEncoding("GB2312");
        // SMTP 设置
        SmtpClient SmtpC = new SmtpClient(this.Smtp, 25);
        SmtpC.UseDefaultCredentials = false;
        SmtpC.Credentials =
          new System.Net.NetworkCredential(this.UserName, this.Password);
        SmtpC.DeliveryMethod = SmtpDeliveryMethod.Network;
        SmtpC.Send(MailM);             // 发送邮件
    }
    catch (Exception e)
    {
        HttpContext.Current.Response.Write("<script>alert('"
          + e.Message.ToString() + "');</script>");
    }
}
```

经过上面几个步骤，MailHelper 类的基本框架就搭建好了。剩下的工作就是完善并实现相应的功能，如图 13-10 所示为该类最终的结构。

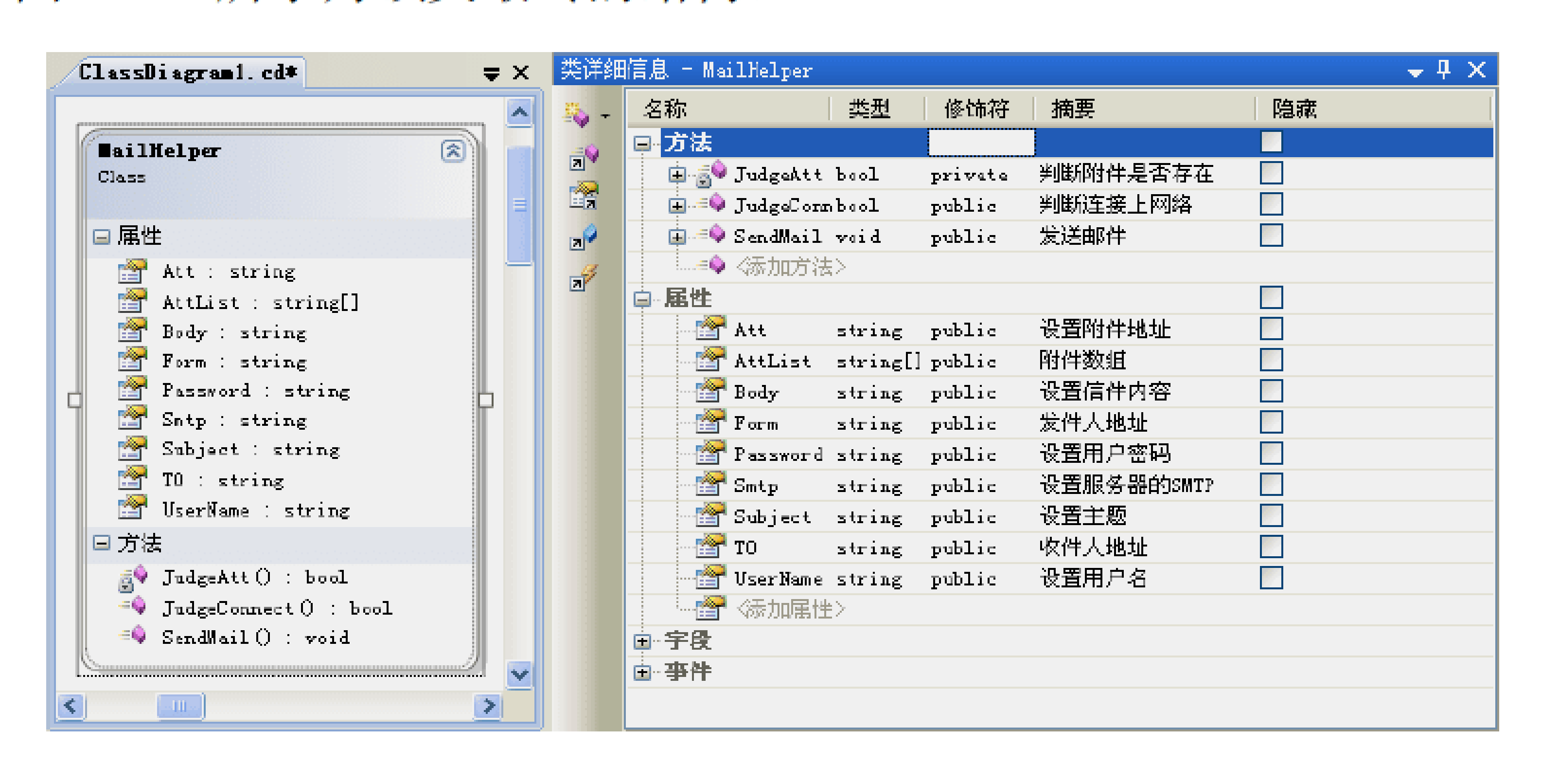


图 13-10　MailHelper类的结构

(6) 新建一个 ASPX 页面，并调用 MailHelper 类来发送邮件。在布局中添加用于输入收件人、邮件主题和邮件正文的文本框，还有“发送”按钮。

(7) 打开“发送”按钮的单击事件，在这里编写详细的邮件发送代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    MailHelper mh = new MailHelper();

    mh.Smtp = "smtp.163.com";           //SMTP 服务器
    mh.UserName = "ayhncn";             //登录账户
    mh.Password = "163.com";            //登录密码
    mh.Form = "ayhncn@163.com";         //发送者邮箱
    mh.Subject = txtSubject.Text.Trim();        //邮件主题
    mh.Body = txtBody.Text;             //邮件内容
    mh.TO = txtTo.Text.Trim();          //收件人

    if (mh.JudgeConnect())              //网络是否正常
    {
        ltStatus.Text = "网络状态正常。正在发送...";
        try
        {
            mh.SendMail();              //发送邮件
            ltStatus.Text = "发送成功。";
            Page.ClientScript.RegisterStartupScript(
              this.GetType(), "", "<script>alert('发送成功。');</script>");
        }
        catch (System.Net.Mail.SmtpException ex)
        {
            Page.ClientScript.RegisterStartupScript(this.GetType(), "",
              "<script>alert('" + ex.Message.ToString() + "');</script>");
        }
    }
    else
    {
        ltStatus.Text = "网络连接故障，请检查后重试。";
    }
}
```

可以看出，与邮件发送有关的设置都在程序中指定，用户只需填写三项即可。代码中的 ltStatus 是一个 Literal 控件，用于显示状态信息。

至此，本实例的发送邮件就制作完成了。

但是为了更加实用，可以将输入邮件内容的控件替换为第三方文本编辑器，例如这里使用的是 kindeditor。



作者在这里使用的是 163 的 SMTP 服务器，不同服务器的地址和端口可能有所不同。即使同一 SMTP 服务器，受账户和密码的限制也可能出现不同的结果。

Web 程序员成功之路

13.4.4　运行结果

从项目中运行带有发送邮件的页面，然后填写邮件的各个选项。完成之后，单击“发送”按钮开始发送邮件的过程。如果成功，将看到如图 13-11 所示的效果。

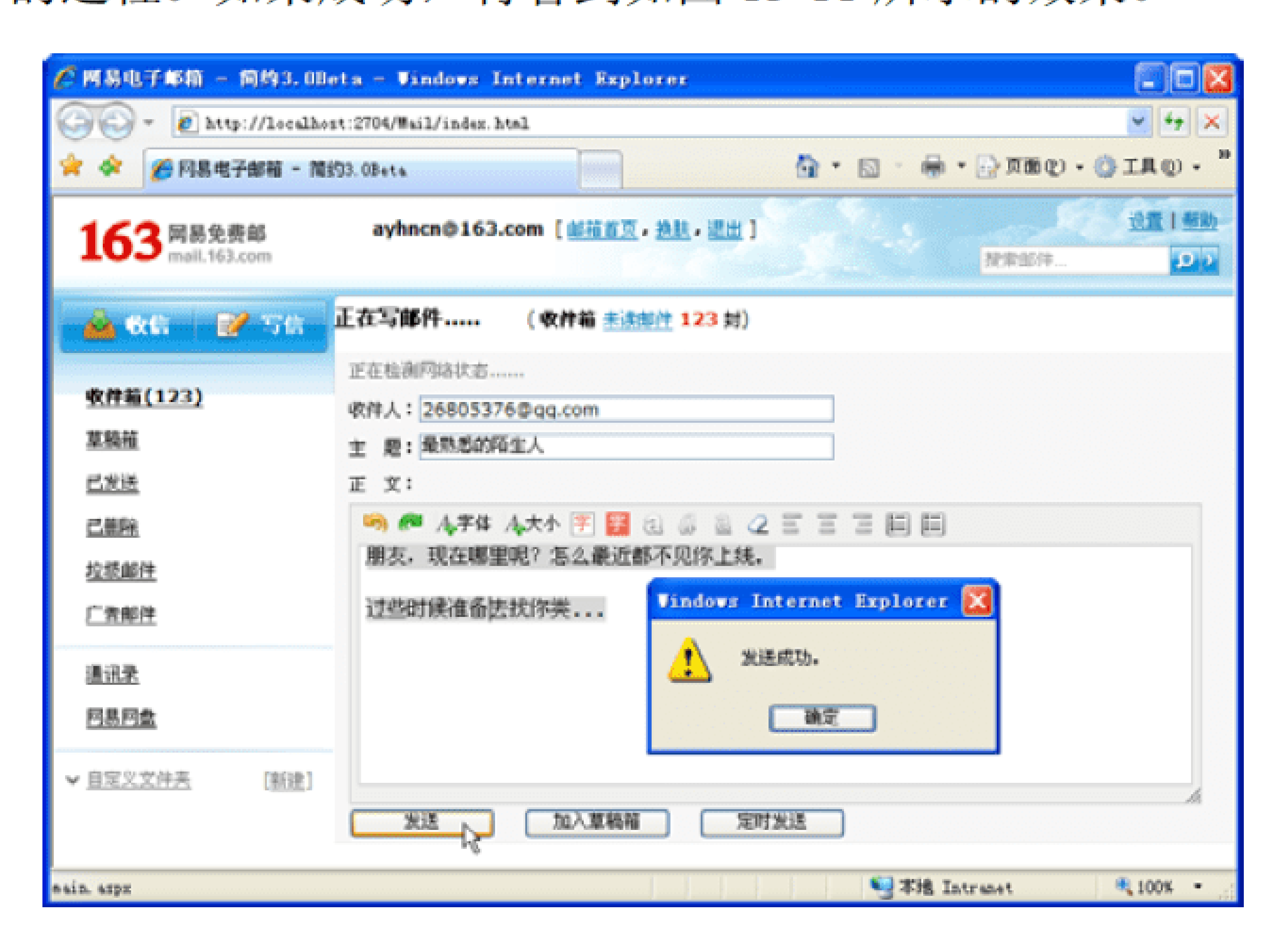


图 13-11　发送邮件的效果

此时，打开收件人的邮箱，将会看到一封未读邮件，其中的内容正是在上面编写的，说明发送成功，如图 13-12 所示。



图 13-12　查收邮件的效果

13.4.5　实例分析



源码解析：

可以看到，在强大的 ASP.NET 下实现邮件的发送是多么简单。

首先设置好 SMTP 服务器信息，之后便可以由用户来填写邮件的内容。接下来通过封装在 MailHelper 类中的 SendMail()方法，将这些信息组织成一个邮件格式并发送出去。

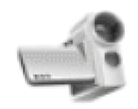
需要注意的是：身份验证用到的账号是网页登录时用到的账号，如果你的邮箱地址是abc@163.com，在登录 mail.163.com 邮箱的时候，输入的账号是 abc 而不是 abc@163.com，身份验证的时候也是如此。

13.5 使用ASP.NET第三方组件

不少应用程序的作者为编程人员提供了能在其他程序中复用的组件。我们通常会在自己的程序中引入这些第三方组件，再调用提供的方法来实现相对复杂的功能。在本节中精选了常用3 个组件进行介绍。

13.5.1 分页组件

分页是 Web 应用程序中最常用到的功能之一。选择一种简洁、高效的处理方式，既可以加快程序运行速度，也提高了开发质量。AspNetPager 正是基于这种思想而设计的一款分页组件，它简单、实用，在项目后台中的表现效果也非常棒！



视频教学：光盘/videos/13/AspNetPager.avi　　　长度：11 分钟

1. 基础知识——AspNetPager

AspNetPager 提出了与众不同的 ASP.NET 分页问题解决方案，即将分页导航功能与数据显示功能完全独立开来，由用户自己控制数据的获取及显示方式。因此可以被灵活地应用于任何需要实现分页导航功能的地方，例如为 GridView、DataList 以及 Repeater 等数据绑定控件实现分页。AspNetPager 的官方网站是 http://www.webdiyer.com，目前最新版本为 7.3.2，下面以此版本为例介绍使用方法。

下载带示例的项目，然后在 VS2008 中以打开网站的方式定位到解压后的文件夹。在 bin 目录下有一个 AspNetPager.dll 文件，这便是该组件的程序集文件。

直接运行网站，便可以打开附带的完整帮助文档及示例项目，能够帮助用户快速上手，熟悉 AspNetPager 组件的使用。如图 13-13 所示即为查看示例的效果。

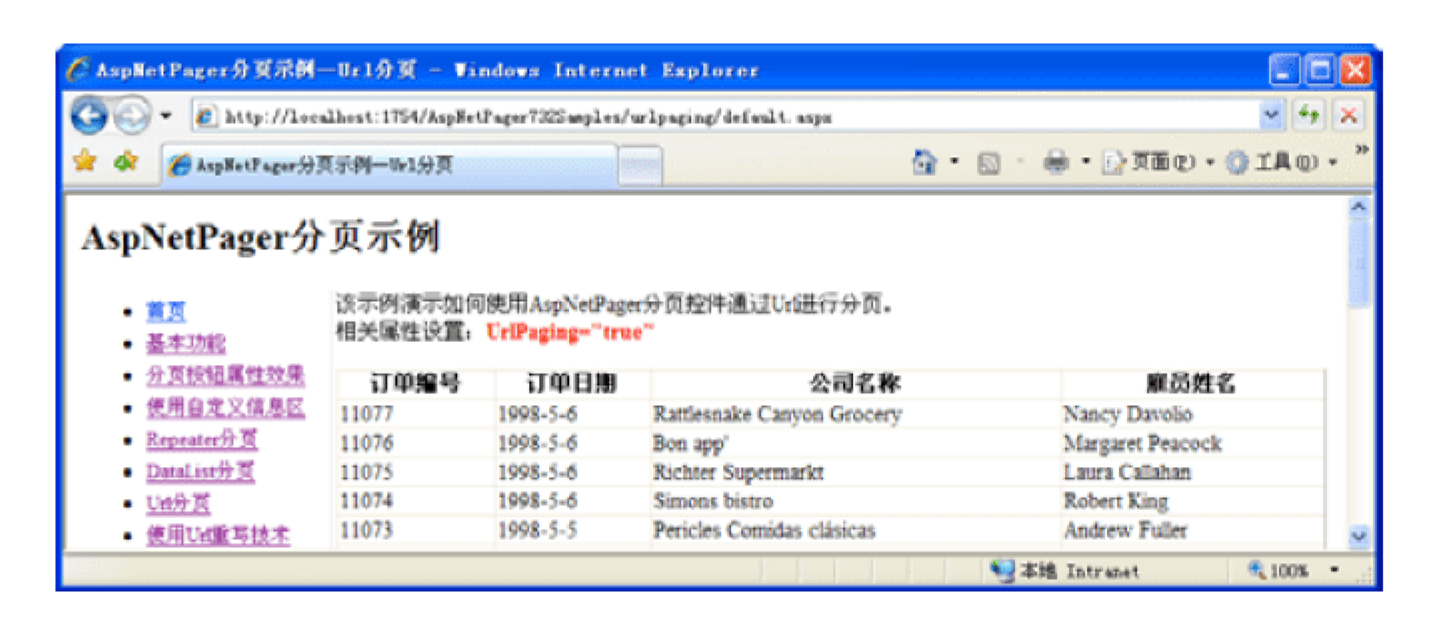


图 13-13　AspNetPager组件的示例效果

如果希望在一个新项目中使用 AspNetPager 组件，方法也很简单。只需在项目的“工具箱”中添加对 AspNetPager.dll 文件的引用，然后与普通的控件一样，拖动到页面上即可，效果如

图 13-14 所示。

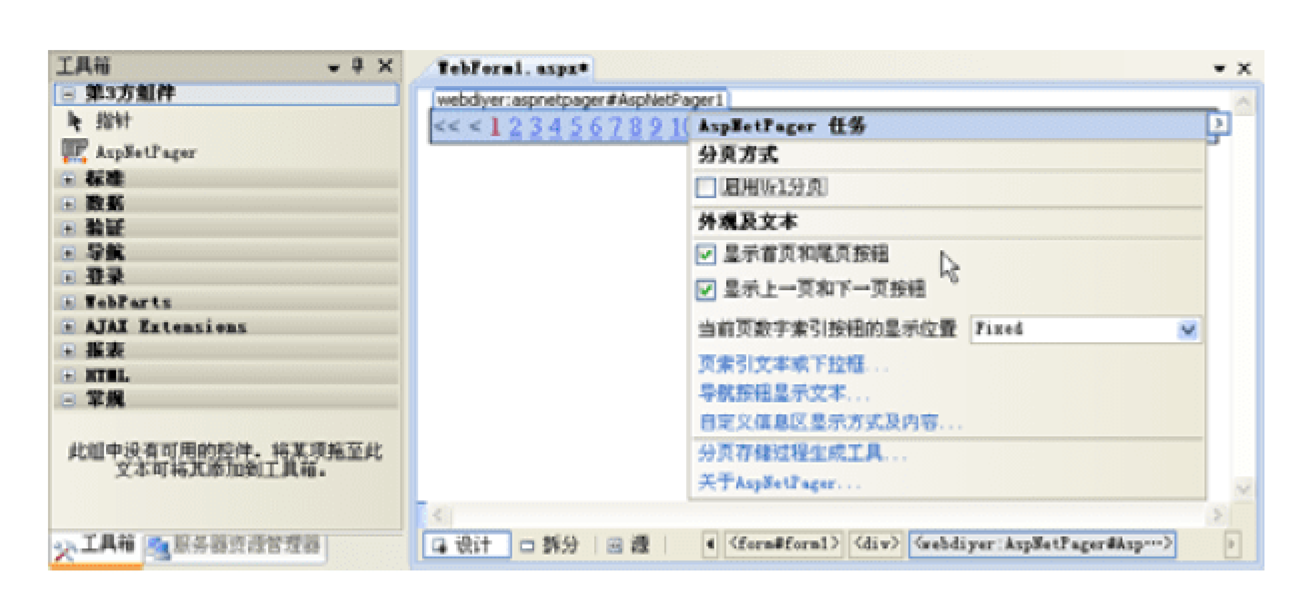


图 13-14　添加AspNetPager组件

2. 实例描述

使用 ASP.NET 中的 DataGrid(ASP.NET 1.1)和 GridView(ASP.NET 2.0)控件可以实现分页功能，但其分页功能并不尽如人意，如可定制性差、无法通过 URL 实现分页功能等。

虽然 ASP.NET 3.5 中提供了一个单独的分页组件 DataPager，仍然没有从本质上改变分页的性能，编写时技术难度大、任务繁琐而且代码重用率极低，因此分页已成为许多 ASP.NET 程序员最头疼的问题之一。

AspNetPager 组件的出现，一举突破了众多难题，成为使用频率最高的第三方组件。下面我们就来看看它是如何与 Repeater 组件结合实现分页的。

3. 实例应用

【例 13-5】实现分页显示。

(1) 在网站中添加 AspNetPager 组件。

(2) 使用 Repeater 控件实现获取数据库的数据并绑定显示，显示布局效果如图 13-15 所示。



图 13-15　Repeater控件显示布局

(3) 从“工具箱”中将 AspNetPager 组件放在 Repeater 控件的下方。然后在“源”视图中对分页显示的属性进行设置，最终如下所示：

```
<webdiyer:AspNetPager id="AspNetPager1" runat="server" PageSize="10"
  AlwaysShow="True" ShowNavigationToolTip="true"
  PageIndexBoxType="DropDownList"
  ShowCustomInfoSection="Left" ShowPageIndexBox="always"
  CustomInfoHTML="第<font color='red'><b>%currentPageIndex%</b></font>页，共
    %PageCount%页，每页显示%PageSize%条记录"
  OnPageChanged="AspNetPager1_PageChanged">
</webdiyer:AspNetPager>
```

下面对上述代码中使用的 AspNetPager 组件属性和事件进行简单介绍，如表 13-1 所示。

表 13-1　AspNetPager组件的常用成员

名　称	说　明
PageSize 属性	每页显示的行数
AlwaysShow 属性	指定是否总是显示 AspNetPager 分页组件，即使要分页的数据只有一页
ShowNavigationToolTip 属性	指定当鼠标指针悬停在导航按钮上时是否显示工具提示
PageIndexBoxType 属性	指定页索引框的显示类型，可以是允许用户手工输入的文本框和只能选择的下拉框。可选值有 TextBox 和 DropDownList
ShowCustomInfoSection 属性	指定显示用户自定义信息区的方式
ShowPageIndexBox 属性	指定页索引框的显示方式，以便用户输入或从下拉框中选择需要跳转到的页索引。可选值有 Never、Auto 和 Always
CustomInfoHTML 属性	指定显示在用户自定义信息区的用户自定义 HTML 文本内容
OnPageChanged 事件	当页面发生变化，即页面切换时触发该事件，属于分页时必需的事件

(4) 对先前使用 Repeater 控件实现的数据显示添加分页功能。修改 Page_Load()为如下的代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        AspNetPager1.RecordCount =
          GetDataCounts(); //将记录总数赋予 RecordCount 属性
        bindData();       //绑定数据
    }
}
```

上述代码中使用了 AspNetPager 的 RecordCount 属性，该属性用于获取或设置需要分页的所有记录的总数。该属性的值由自定义方法 GetDataCounts()返回。bindData()方法则比较简单，用来绑定数据到 Repeater 控件。

(5) 在前台中定义了 PageChanged 事件。这一步编写它的处理方法，代码如下：

```
protected void AspNetPager1_PageChanged(object sender, EventArgs e)
{
    bindData();  //绑定数据
}
```

(6) 最后我们来看看 bindData()中是如何获取指定位置数据的。最终代码如下：

```
void bindData()
{
    int startIndex =
      AspNetPager1.StartRecordIndex; //获取当前页数据记录的起始索引
    int endIndex =
      AspNetPager1.EndRecordIndex;  //获取当前页数据记录的结束索引
```

```
    string strSql =
      "select 图书编号,图书名称,出版社,定价,日期,是否借出 from 图书明细";
    DataSet ds = ExecuteSql(strSql);                     //获取所有数据
    PagedDataSource pds = new PagedDataSource();
    pds.AllowPaging = true;                              //启用分页
    pds.PageSize = AspNetPager1.PageSize;                //指定每页显示数量
    pds.CurrentPageIndex = AspNetPager1.CurrentPageIndex - 1; //对数据进行分页
    DataView dv = ds.Tables[0].DefaultView;
    pds.DataSource = dv;
    Repeater1.DataSource = pds;                  //将分页后的数据指定为数据源
    Repeater1.DataBind();                        //绑定显示
}
```

(7) 在上步中的 ExecuteSQL()为一个通用的方法，参数为要执行的查询语句，返回执行后的 DataSet。具体实现代码这里不再给出，读者可参考源文件。

4. 运行结果

运行带有 Repeater 和 AspNetPager 的页面，可以看到默认会显示 10 条数据作为第 1 页。在底部为一个分页栏，显示了当前页、总页数、每页数量、页导航以及跳转列表。

当把鼠标移到页的导航上时会显示提示，这是由 ShowNavigationToolTip 属性控制的，如图 13-16 所示。



图 13-16　分页显示效果

5. 实例分析



源码解析：

AspNetPager 是一个扩展性极强的组件，最简化时不用一行代码，只需设置一个属性即可实现分页。

本实例中对使用它与 Repeater 控件的结合做了重点介绍，这一过程中涉及了该组件的基本功能。如设置分页显示的属性、添加分页事件 PageChanged、获取指定页的数据等。

AspNetPager 组件的 StartRecordIndex 和 EndRecordIndex 属性在自定义分页时非常有用。

它可以取到当前页中数据的起止索引范围，根据这个设置数据源。

最后说明一下，AspNetPager 控件和数据是相互独立的，因此要分页的数据可以来自任何数据源。

13.5.2 实现未读邮件的提示对话框

这里向读者推荐一个可以实现类似 MSN 的通知消息窗口效果的.NET 组件——PopupWin。它的出现改变了以前编写繁琐 JavaScript 代码的情况，可以有像 MSN 一样的那种消息弹出提示框效果，而且支持多种弹出效果，同时也可以设置事件等。

 视频教学：光盘/videos/13/PopupWin.avi　　长度：7 分钟

1. 基础知识——PopupWin

PopupWin 是由国外一个牛人开发的、适用于 ASP.NET 的第三方组件。它可以很容易地实现各种消息弹出功能，而且内置了多种皮肤和动画方案，还能够控制显示的位置以及时机。

我们可以通过地址 http://www.codeproject.com/KB/custom-controls/asppopup.aspx 下载得到 PopupWin 组件，以及该组件的示例项目。

将下载项目 bin 目录下的 EeekSoft.Web.PopupWin.dll 文件添加到 VS2008 的工具箱中，我们会看到多出了 PoPupWinAnchor 组件和 PopupWin 组件。每个 PopupWin 组件表示一个弹出面板，PoPupWinAnchor 组件则表示弹出面板的管理器。

向页面添加一个 PopupWin 组件，它会自动在页面顶部添加相关引用，示例代码如下：

```
<%@ Register assembly="EeekSoft.Web.PopupWin" namespace="EeekSoft.Web"
  tagprefix="cc1" %>
```

然后利用它的属性控制弹出面板的外观，示例代码如下：

```
<cc1:PopupWin ID="PopupWin1" runat="server" ColorStyle="Blue"
  Width="230px" Height="100px" DockMode="BottomLeft" WindowScroll="false"
  WindowSize="300,200" />
```

至此，PopupWin 组件的使用就算完成了，非常简单。运行时会看到从页面左下部向上弹出的一个面板，效果非常棒。

PopupWin 组件有两种事件可以被激发：Linkcliked 事件(当消息框内的连接被点击时触发)和 Popupclosed 事件(当消息框窗口被关闭时触发)。

它有三种方式对这些事件进行处理，而 ActionType 属性的取值，则决定了这三种方式。

- messagewindow：默认的弹出窗口方式，将以设置好的 Title 属性和 Text 属性为标题和窗口内的文字说明。
- openlink：此时，控件允许当点击窗口内的文字链接时，将以打开新窗口的方式打开该链接。
- raiseevenst：当选择该属性时，控件将会在服务端处理 linkcliked 事件和 popupclosed 事件。

2. 实例描述

现在很多的客户端软件都有这样一个非常实用的功能：程序运行后会自动地在系统托盘的右下角由下往上弹出消息框来通知一些信息，如 QQ 好友上线、杀毒情况或者最新升级版本等。

同样在论坛里，当有新回复的帖子时，系统会自动弹出消息提示框，又或者在一个电子政务的系统里，当收到新的邮件或者工作单时，可以使系统弹出消息框提示等。

那么，在 ASP.NET 开发的 Web 应用程序中，如何实现这样的功能呢？下面我们就来看看如何用 PopupWin 组件在 ASP.NET 中实现有未读邮件时的提示对话框。

3. 实例应用

【例 13-6】实现未读邮件的提示对话框。

(1) 打开我们的邮件项目，然后将 PopupWin 组件添加到工具箱。

(2) 进入主界面，添加 ScriptManager 和 UpdatePanel 控件，使页面不刷新。

(3) 在 UpdatePanel 内添加一个 PopupWin 组件，并调整其属性，最终为如下代码：

```
<cc1:PopupWin id="popupWin" runat="server"
  width="230px" height="100px" windowsize="330, 180"
  windowscroll="False" dockmode="BottomLeft" colorstyle="Blue"
  gradientdark="210, 200, 220" textcolor="0, 0, 3" shadow="125, 90, 160"
  lightshadow="185, 170, 200" darkshadow="128, 0, 102" visible="False"
  showlink="True" >
</cc1:PopupWin>
```

(4) 默认情况下，仅在页面加载完成后显示一次面板。这里为了方便多次查看，在上述控件的下方添加一个 PoPupWinAnchor 组件，不用做任何属性设置。

(5) 在页面的合适位置添加如下代码来控制多次查看的链接：

```
<span id="spanReopen" runat="server">重新显示</span >
```

(6) 如果现在运行，虽然可以看到弹出面板的提示效果，但内容和链接都无法设置。这就需要在 Page_Load()事件中对这些信息进行预定义，代码如下所示：

```
protected void Page_Load(object sender, EventArgs e)
{
    PopupWinAnchor1.HandledEvent = "onclick";        //指定重新显示的事件
    PopupWinAnchor1.LinkedControl = "spanReopen";    //指定目标 ID
    PopupWinAnchor1.PopupToShow = "popupWin";        //指定面板 ID
    popupWin.HideAfter = (sender == null) ? -1 : 5000;
    popupWin.Visible = true;                         //使面板可见
    popupWin.Message = "你的邮箱中有 2 封未读邮件。";    //定义内容，支持 HTML
    popupWin.Text =
      "用户 somboy,你上次登录时间是 2010-09-15。<br/>您有 2 封未读邮件快去看看吧。";
                                   //定义单击后的详细内容，支持 HTML
    popupWin.DragDrop = false;             //不可拖动
    popupWin.DockMode = EeekSoft.Web.PopupDocking.BottomRight;
    popupWin.ColorStyle = EeekSoft.Web.PopupColorStyle.Green;
    popupWin.Title = "新邮件";             //设置面板的标题
}
```

(7) 至此，为邮件项目添加带未读邮件提示的功能就制作完成了。

4. 运行结果

从邮件项目中选择上面修改的页面并运行。打开后，会在页面的右下角逐渐弹出一个新邮件的提示面板，效果如图 13-17 所示。

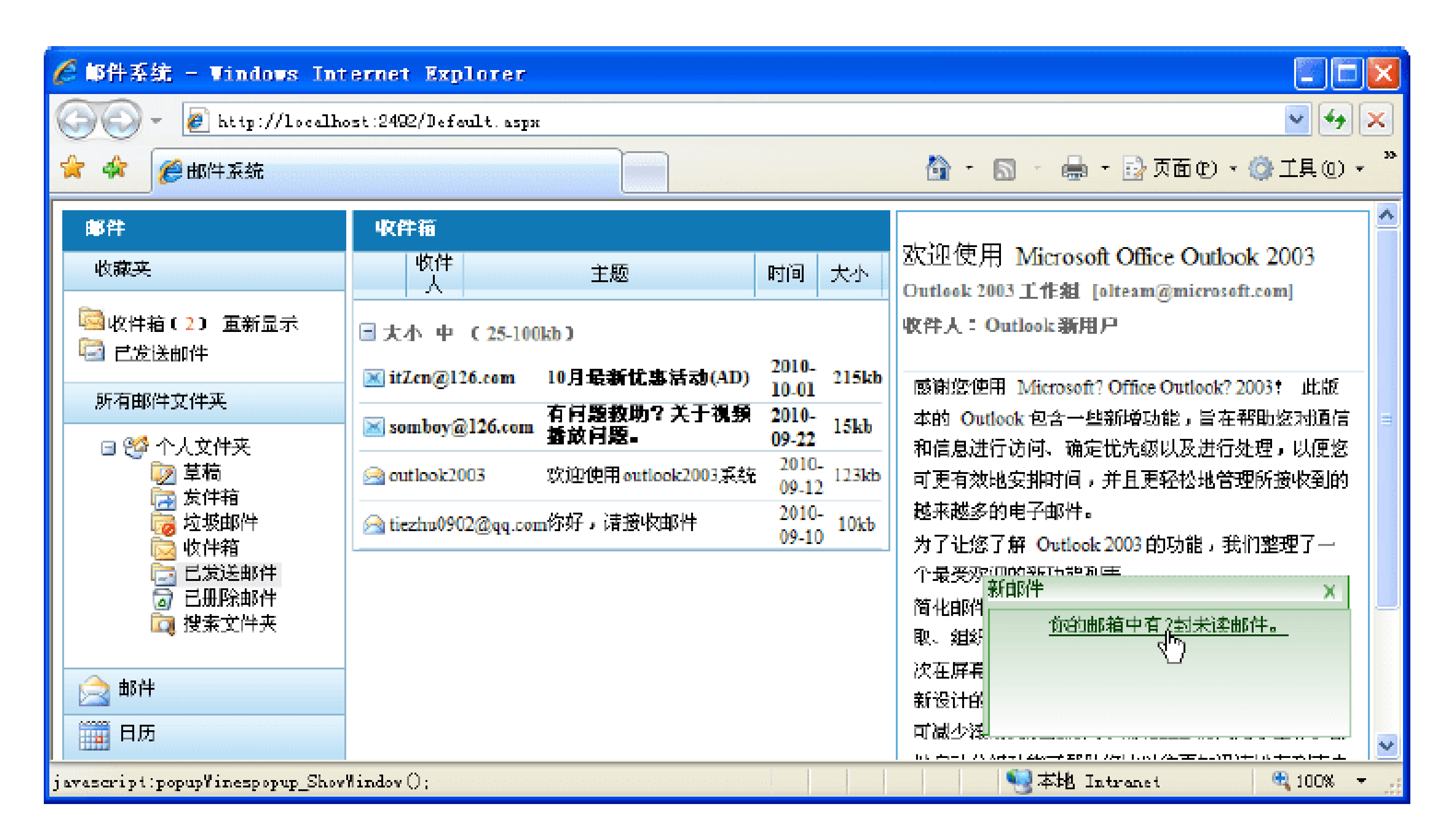


图 13-17 运行时新邮件的提示效果

单击其中的超链接，可以在弹出的窗口查看详细内容，效果如图 13-18 所示。

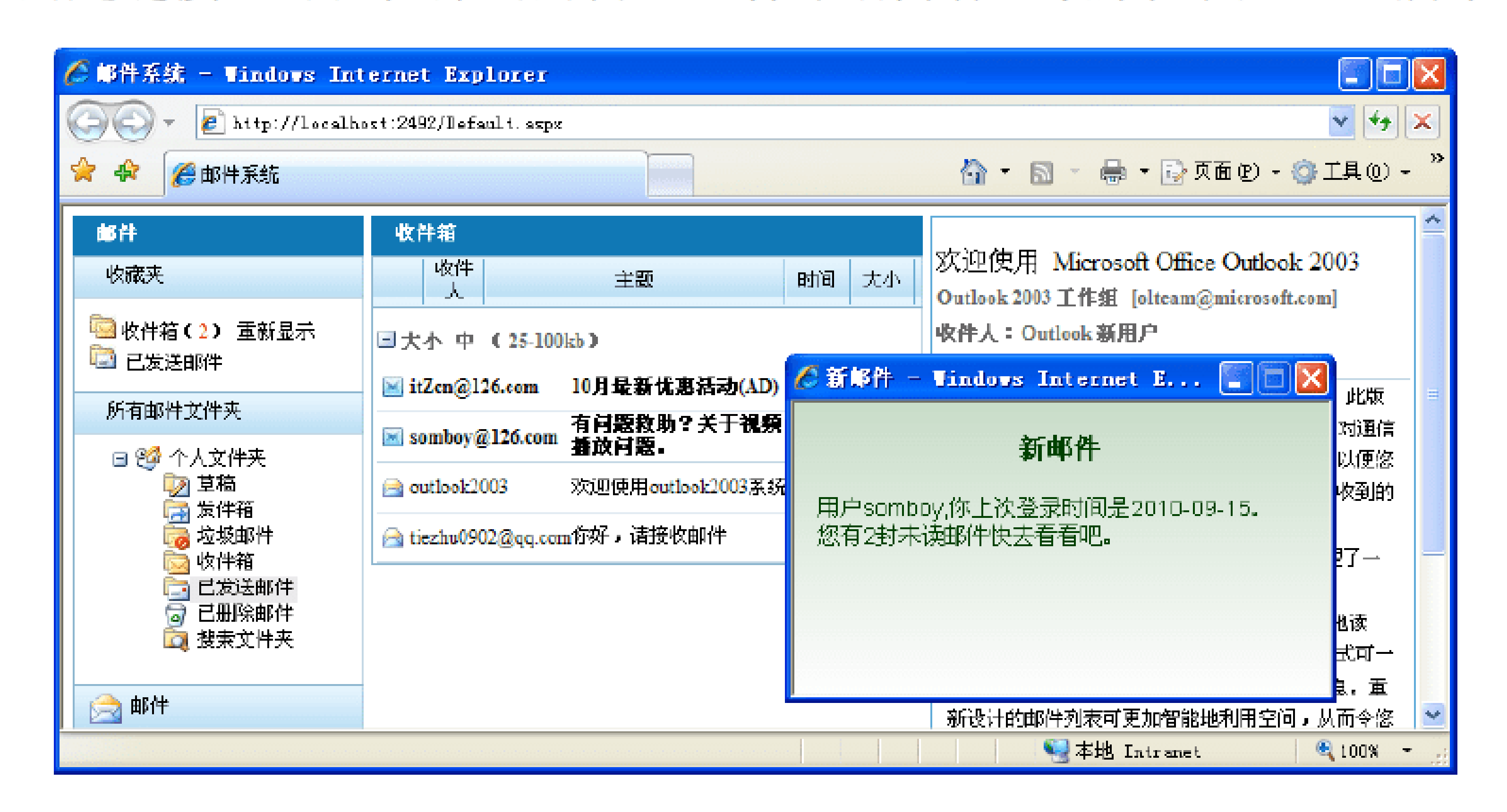


图 13-18 查看提示内容的效果

5. 实例分析



源码解析：

通过本实例中的几个简单步骤体验了 PopupWin 组件的使用过程。

该组件提供了大量属性来由用户自定义提示框的外观，包括显示的宽高、弹出窗口的宽高、弹出位置、阴影和字体颜色等。

同时该组件还有很多方法和事件。本实例中前台主要设置显示的外观效果，在后台中通过代码设置显示的内容。实际应用时，这些内容可能来自数据库。

13.5.3　日志记录组件

log4net 是.NET 下一个非常优秀的开源日志记录组件。log4net 记录日志的功能非常强大，它可以将日志分不同的等级，以不同的格式输出到不同的媒介。

 **视频教学：光盘/videos/13/log4net.avi**　　长度：9 分钟

1. 基础知识——log4net

log4net 组件主要由 5 部分组成，分别为 Appenders、Filters、Layouts、Loggers 和 Object Renders。这 5 个组成部分的主要功能如下。

- Appenders：用来定义日志的输出方式。
- Filters：用来定义过滤器，可以过滤掉 Appender 输出的内容。
- Layouts：用于控制 Appender 的输出格式，可以是线性的，也可以是 XML 的。但一个 Appender 只能有一个 Layout。
- Loggers：直接与应用程序交互的组件。Logger 只是产生日志，然后由它引用的 Appender 记录到指定的媒介，并由 Layout 控制输出格式。
- Object Renders：它将告诉 Logger 如何把一个对象转化为一个字符串记录到日志里。

要想在我们的项目中能够使用 log4net 组件，需要引用 log4net.dll 文件，这个文件可以从官方站下载 http://logging.apache.org/log4net/download.html。然后在 ASP.NET 项目中添加对 log4net.dll 的引用，就可以在程序中使用了。

在 AssemblyInfo.cs 文件中还需要添加一行代码来初始化配置，并监测配置文件的变化，一旦发生修改，将自动刷新配置。代码如下：

```
[assembly: log4net.Config.XmlConfigurator()]
```

最后，在 Web.config 中对日志记录格式进行配置。如下所示为一个示例配置信息：

```
<configuration>
<configSections>
    <section name="log4net"
      type="System.Configuration.IgnoreSectionHandler"/>
</configSections>
<log4net debug="true">
    <appender name="RollingLogFileAppender"
      type="log4net.Appender.RollingFileAppender">
    <file value="d:\test.log" />
    <appendToFile value="true" />
    <rollingStyle value="Size" />
    <maxSizeRollBackups value="10" />
    <maximumFileSize value="2KB" />
    <staticLogFileName value="true" />
    <layout type="log4net.Layout.PatternLayout">
    <footer value="[Footer]—汇智科技 &#13;&#10;" />
    <conversionPattern value="记录时间：%date ,线程 ID:[%thread],
```

```
    日志级别：%-5level，信息描述：%message%newline" />
   </layout>
   </appender>
   <root>
   <priority value="ALL" />
   <appender-ref ref="RollingLogFileAppender" />
   </root>
</log4net>
```

2. 实例描述

在前面的小节中，我们讲到防止用户重复登录时引用了一个小故事。现在想想，如果每个会员在登录的时候，站长的系统对每个会员的登录信息都有日志记录的话，最后，发现问题的时候一看日志也就一目了然了。嗯，其实日志的作用和好处远不止这些。

下面我们就题论题，针对我们前面讲的防止用户重复登录的实例，给该实例再添加一个记录用户登录信息的功能。

3. 实例应用

【例 13-7】使用第三方组件 log4net 记录登录信息。

(1) 在 VS2008 中打开“防止用户重新登录”的项目，在该项目中我们按照基础知识中的步骤来配置 log4net 组件，在这里我们将用户的日志文件保存到 D 盘根目录下，命名为“test.log”。

(2) 在登录页面的后台代码设计界面，我们按照需求修改代码，详细代码如下：

```
//创建日志对象
private static readonly ILog log = LogManager.GetLogger(typeof(login).Name);

protected void Page_Load(object sender, EventArgs e)
{
   log.Info("登录页面加载成功");            //记录页面信息
}

protected void loginBtn_Click(object sender, ImageClickEventArgs e)
{
   string name = this.username.Text.Trim();
   string pwd = this.userpwd.Text.Trim();
   log.Info("用户" + name + "开始登录主页");   //记录用户信息
   string con =
     ConfigurationManager.ConnectionStrings["con"].ConnectionString;
   string sqlStr = string.Format("select count(*) from UserTable
                    where username='{0}' and userpwd='{1}'", name, pwd);
   SqlConnection conn = new SqlConnection(con);
   conn.Open();
   SqlCommand cmd = new SqlCommand(sqlStr, conn);
   int n = Convert.ToInt32(cmd.ExecuteScalar());
   if (n > 0)
   {
      log.Info("用户" + name + "验证成功");
```

```
        ArrayList list;
        list = Application.Get("User_list") as ArrayList;
        if (list == null)
        {
            list = new ArrayList();
        }
        else
        {
            for (int i=0; i<list.Count; i++)
            {
                if (name==list[i].ToString())
                {
                    ClientScript.RegisterStartupScript(this.GetType(), "",
                      "<script>alert('该用户:" + name + "已登录! ')</script>");
                    //记录用户信息
                    log.Info("用户" + name + "已经登录，页面仍留在登录界面");
                    list.Remove(name);
                    Application.Add("User_list", list);
                }
            }
        }
    }

    Session["name"] = name;
    list.Add(name);
    Application.Add("User_list", list);

    ClientScript.RegisterStartupScript(this.GetType(), "",
                      "<script>location.href = 'index.aspx'</script>");
    else
    {
        ClientScript.RegisterStartupScript(this.GetType(), "",
                      "<script>alert('用户名或密码错误! ')</script>");
        log.Info("用户" + name + "输入的信息有误，验证失败! ");    //记录用户信息
    }
}
```

(3) 至此，login.aspx 页面中的代码我们修改完了。下面根据以上代码，再来适当地修改一下 YHTop.aspx 页面。这步的实际操作与步骤 2 大同小异，这里不再具体给出代码，请读者参考本节的源文件。

4. 运行结果

当以上页面都修改完后，我们来运行一下程序，看看日志功能是否实现了。在经过一番用户登录、安全退出、重复登录、多用户登录等操作后，我们打开 D 盘下的日志文件，该文件确实记录了用户每一步的关键操作。

日志文件内容如图 13-19 所示。

```
test.log - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)
记录时间：2010-09-20 16:26:03,531 ,线程ID:[8]，日志级别：INFO ，信息描述：主页加载成功！
记录时间：2010-09-20 16:26:03,546 ,线程ID:[8]，日志级别：INFO ，信息描述：用户admin退出系统！
记录时间：2010-09-20 16:26:03,578 ,线程ID:[8]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:26:08,312 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:26:08,312 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin开始登录主页
记录时间：2010-09-20 16:26:08,312 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin验证成功，开始跳
转到主页
记录时间：2010-09-20 16:26:08,312 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin已经登录，页面仍
留在登录界面
记录时间：2010-09-20 16:26:08,390 ,线程ID:[8]，日志级别：INFO ，信息描述：用户admin登录成功！
记录时间：2010-09-20 16:26:08,390 ,线程ID:[8]，日志级别：INFO ，信息描述：主页加载成功！
记录时间：2010-09-20 16:26:21,656 ,线程ID:[4]，日志级别：INFO ，信息描述：用户aa登录成功！
记录时间：2010-09-20 16:26:21,656 ,线程ID:[4]，日志级别：INFO ，信息描述：主页加载成功！
记录时间：2010-09-20 16:26:21,656 ,线程ID:[4]，日志级别：INFO ，信息描述：用户aa退出系统！
记录时间：2010-09-20 16:26:21,687 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:26:26,578 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin登录成功！
记录时间：2010-09-20 16:26:26,578 ,线程ID:[4]，日志级别：INFO ，信息描述：主页加载成功！
记录时间：2010-09-20 16:26:26,593 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin退出系统！
记录时间：2010-09-20 16:26:26,609 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
[Footer]--汇智科技。
记录时间：2010-09-20 16:30:26,062 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:30:31,390 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:30:31,390 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin开始登录主页
记录时间：2010-09-20 16:30:31,421 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin验证成功，开始跳
转到主页
记录时间：2010-09-20 16:30:31,578 ,线程ID:[4]，日志级别：INFO ，信息描述：用户admin登录成功！
记录时间：2010-09-20 16:30:31,578 ,线程ID:[4]，日志级别：INFO ，信息描述：主页加载成功！
记录时间：2010-09-20 16:30:36,046 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
记录时间：2010-09-20 16:30:39,734 ,线程ID:[4]，日志级别：INFO ，信息描述：登录页面加载成功
```

图 13-19　日志文件的内容

5. 实例分析



源码解析：

本实例的源码中，我们并没有对原项目做太多的修改，只是在页面的适当位置加入了下面这句代码：

```
log.Info("内容为记录到日志中的信息");
```

然而，仅靠简单的一句代码，是不能完成日志功能的，主要应归功于第三方组件 log4net。另外，在该组件使用的过程中我们一定要先创建日志实例，代码如下：

```
private static readonly ILog log = LogManager.GetLogger(typeof(login).Name);
```

13.6　常见问题解答

13.6.1　ASP.NET数据绑定



ASP.NET 数据绑定。

网络课堂：http://bbs.itzcn.com/thread-9873-1-1.html

```
/traditional opera 应用程序中的服务器错误。
--------------------------------------------------------------------------
分析器错误
```

说明：在分析向此请求提供服务所需资源时出错。请检查下列特定分析错误详细信息并适当地修改源文件。

分析器错误消息：未能加载文件或程序集 AspNetPager 或它的某一个依赖项。系统找不到

指定的文件。源错误：

```
行 1: <%@ Register TagPrefix="webdiyer" Namespace="Wuqi.Webdiyer"
Assembly="AspNetPager" %>
行 2: <%@ Page validateRequest="false" language="c#" Inherits="book._index"
CodeFile="index.aspx.cs" %>
源文件: /traditional opera/dongtai/index.aspx  行: 1
```

程序集加载跟踪：下列信息有助于确定程序集 AspNetPager 无法加载的原因。

……

【解决办法】：这个错误是你使用了 AspNetPager 分页控件，这是一个第三方控件，使用的时候需要添加类库文件。你找一个添加进项目就行了。

13.6.2　ASP.NET 2.0(C#)的Web Service是什么



ASP.NET2.0(c#)的 Web Service 是什么，起什么作用？

网络课堂：http://bbs.itzcn.com/thread-9875-1-1.html

ASP.NET 2.0(C#)的 Web Service 是什么，起什么作用？比如我要从一个远程数据库验证用户名密码是否正确，用这个的话是不是不需要知道远程数据库的用户名和密码？

【解决办法】：关于“是什么”，这个做 Web Service 服务的人需要知道，调用的话就不需要知道了，只需要知道它的服务地址在哪，提供了哪些服务，需要输入什么参数，返回的又是一个什么结果即可。一般都可以将所有的业务逻辑封装到 Web Service 里，在远程直接调就可以了，很方便，而且透明。

13.7　习　　题

一、填空题

(1) 假设在页面上有一个 ID 为 pnlMain 的容器，完善下面的代码，实现向容器中添加一个 ID 为 lblTitle 的 Label 控件，其中显示的文本为“测试”：

```
Label lbl = new Label();
lbl.ID = "__________";          //设置 ID
lbl.Text = "__________";        //设置文本
__________;                     //添加到容器
```

(2) 假设 domainService.com.itzcn 是一个外部 Web Service 的引用名称，在下面的空白处填写__________代码实现调用 Query 方法查询 ayhncn.com 域名是否可用：

```
domainService.com.itzcn domain = new domainService.com.itzcn();
bool isValid = __________ ;
```

(3) 要实现发送邮件的功能，必须借助于使用 System.Net.Mail 命名空间下的__________类来完成。

(4) 通过使用 MailMessages 类的__________属性来设置邮件的发件人，__________属性用于设置收件人。

(5) 完善下面的代码，使用登录账户为 itzcn、登录密码为 000000 来发送 MailM:

```
SmtpClient sc = new SmtpClient("smtp.126.com", 25);
sc.Credentials =
  new System.Net.NetworkCredential("__________", "__________");
__________;      //发送邮件
```

二、选择题

(1) 在 System.Net.Mail 命名空间下的__________类封装了邮件的基本功能。

A. MailMessage　　B. SmtpClient
C. Mailer　　D. WebMail

(2) 下面的哪个属性用来设置邮件的抄送人？__________

A. Bcc　　B. Body　　C. CC　　D. Attachments

(3) AspNetPage 组件的__________事件在页面切换时触发。

A. OnPageIndexChanged　　B. OnPageIndexChanging
C. OnePageChanged　　D. PageChanged

(4) PopupWin 组件的 ActionType 属性可以控制事件的处理方式，下面不属于它取值范围的是__________。

A. messagewindow　　B. openlink
C. raiseevenst　　D. newlink

(5) 使用 log4net 时，__________用来控制日志的输出格式。

A. Layouts　　B. Appenders　　C. Filters　　D. Loggers

三、上机练习

上机练习：带分页的图片浏览程序。

借助于 AspNetPager 组件的强大功能，可以对任何数据源实现分页功能。本次练习要求获取指定目录下的文件数量，以此作为分页依据实现图片浏览程序，运行效果如图 13-20 所示。



图 13-20　运行效果



第 14 章　企业信息管理系统

内容摘要：

企业信息管理系统可以让企业很轻松地管理企业新闻和产品信息，并接收用户在线订单。本章采用 ASP.NET + C# + ADO.NET 技术实现简单的企业信息管理系统功能。这个系统基于.NET Framework 平台，使用 C#语言操作 ASP.NET WebForm 和 ADO.NET。开发工具选择 VS2008，数据库选择 SQL Server 2005，实现了新闻、产品、订单的查询和管理功能。

学习目标：

- 了解 ASP.NET 开发系统的流程
- 理解企业信息管理系统的功能划分和架构设计
- 掌握系统数据库的设计和实现过程
- 掌握公共模块的实现方法
- 掌握三层架构的基本开发方法
- 掌握新闻、产品、会员和订单模块的实现
- 掌握 ASP.NET 读取、显示、查询和更新数据库的方法

14.1 系 统 概 述

随着网络的广泛应用，很多企业开始认识到了互联网对于企业宣传与推广的作用，意识到了合理地利用互联网可以在很大程度上提高企业的竞争力，知道了通过互联网为自己做宣传能很好地树立企业形象、提高企业知名度。

最早的时候，企业一般都是制作静态页面构建企业网站，只简单展示企业及商品。这种方式构建的网站存在许多弊端。

对于企业来说，所关注的是网站的实际功效与运营成本，网站维护和信息发布不应该由专业的人员担任，因此作为一个网站系统，要求具有易用性和功能的完善性。

本系统具有以下几个优点：信息发布便捷容易；数据与界面分离，方便网站界面更新；有较强的安全性，对网站访问管理有权限认证以保证网站的安全性。

14.1.1 系统功能

本系统主要分为后台管理模块和前台用户模块。

(1) 后台管理模块的主要功能如图 14-1 所示。

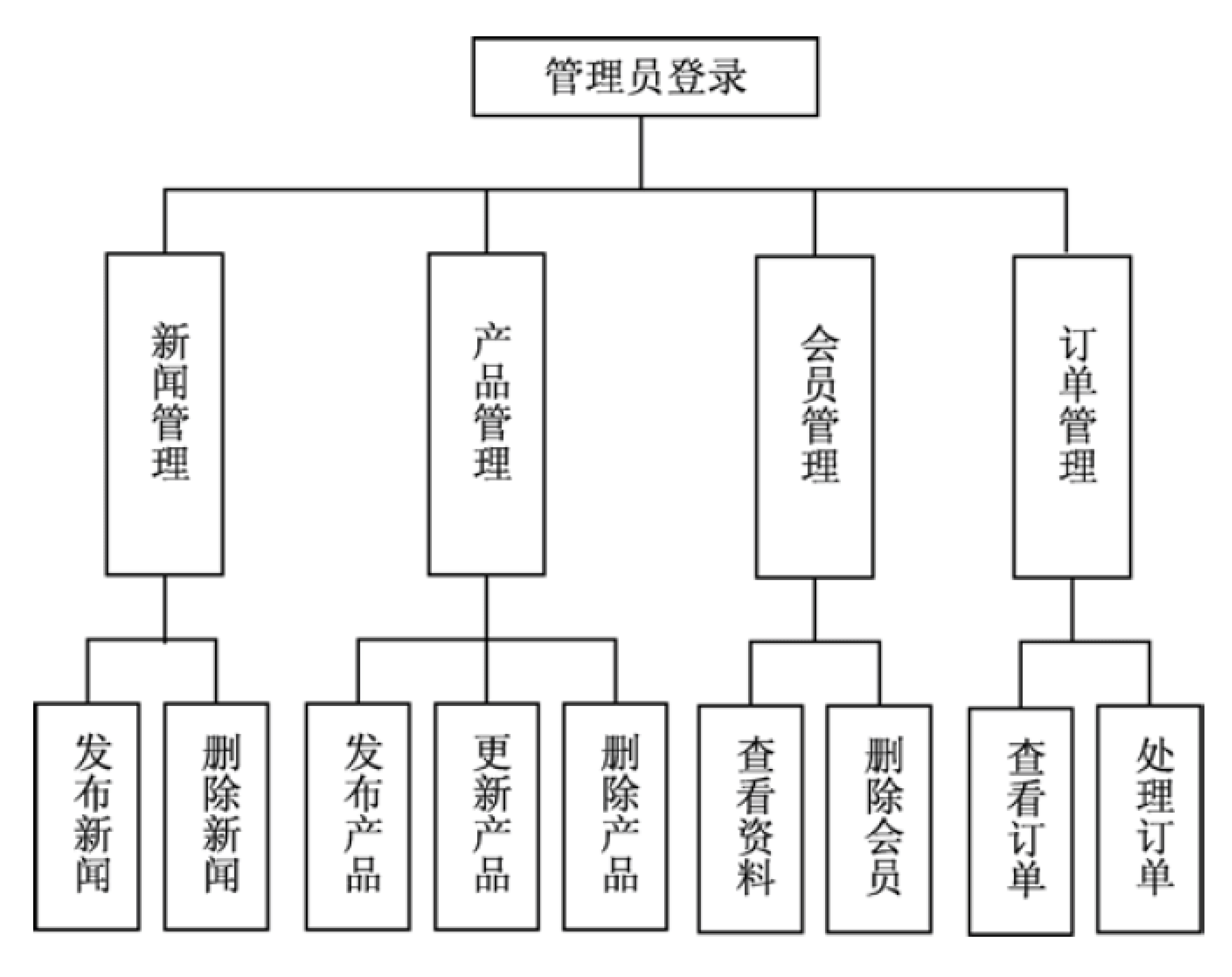


图 14-1 后台管理功能模块

- 新闻管理：新闻管理模块有发布和删除新闻两个功能。为了增强新闻的可读性和易理解性，新闻内容需要编辑样式，插入图片等。
- 产品管理：产品管理模块有发布新产品、修改产品、删除产品功能，为了更好地展现产品信息，产品需要有图片说明。另外，后面要提到的订单功能基于产品，所以删除产品时系统会自动删除该产品关联的订单。

- 会员管理：会员管理模块有查看会员信息和删除会员信息的功能。管理员在这里可以查看会员的所有信息，也可以删除没有价值的会员信息。另外，后面提到的订单功能基于会员，所以删除会员信息时会删除与该会员有关的订单信息。
- 订单管理：订单管理模块有查看订单信息和处理订单两个功能。主要是让管理员在后台能方便地看到订单信息，并对订单做简单的处理。

(2) 前台的主要功能如图 14-2 所示。

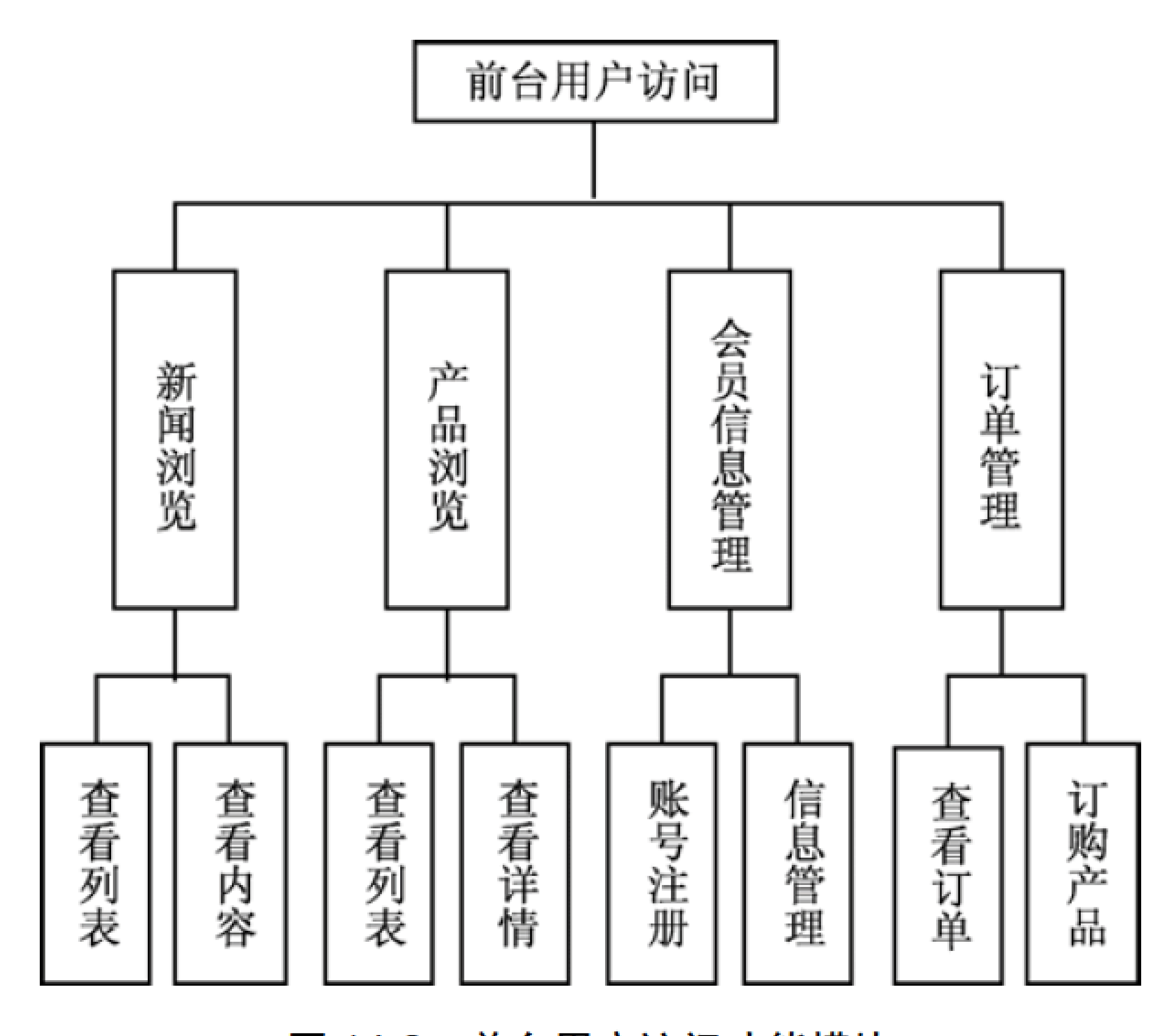


图 14-2　前台用户访问功能模块

- 新闻浏览：访客可以在这里浏览新闻列表，也可以单击新闻标题查看新闻详细内容。
- 产品浏览：访客可以浏览所有产品，也可以查看某个产品的详细信息，如果对某件产品有求购意向，可以在这里单击订购链接订购该产品。
- 个人信息管理：个人信息管理模块提供会员注册、登录、管理个人信息的功能。当用户需要订购产品时，需要填入个人信息注册，成为会员才能进行订购操作。
- 订单管理：前台的订单管理模块能更方便地让会员订购产品和查看自己的订单信息。

14.1.2　系统架构

本系统采用常用的三层架构模式，各司其职，使程序结构清晰，逻辑合理：

- 表现层负责与用户的交互。
- 业务逻辑层封装基本的业务逻辑。
- 数据访问层封装数据库访问方法。

另外加一个“模型层”，实现数据实体的封装。

在“解决方案管理器”中的项目结构如图 14-3 所示。

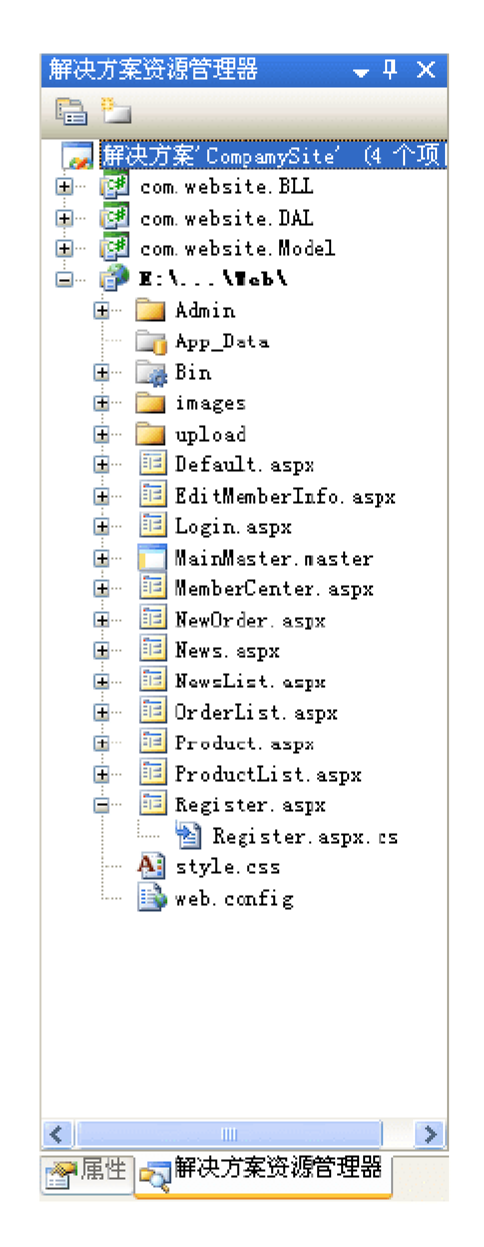


图 14-3　项目解决方案的结构

14.2　数据库的设计和实现

这一节主要讲一下数据库的结构和实现思想。下面针对数据库的分析和设计思路进行详细的描述。

14.2.1　数据库需求分析

通过上述对系统的分析，系统可抽取出以下几类对象：新闻、产品、会员、订单。

这些对象有以下关系。

(1)　会员可以下多个订单。

(2)　一个产品可以被多次订购。

鉴于此，各对象信息至少需要有如下属性：

- 新闻包括标题、发布时间、内容。
- 产品包括名称、图片、详细说明。
- 会员信息包括登录名、密码、性别、年龄、电话、邮箱、地址。
- 订单信息包括产品编号、用户编号、订购数量、发布时间、详细说明、订单状态。

14.2.2　数据库概念结构设计

由上面的分析可知，系统包括以下几个实体对象：新闻、产品、用户信息、订单信息。

实体 E-R 图如图 14-4 所示。

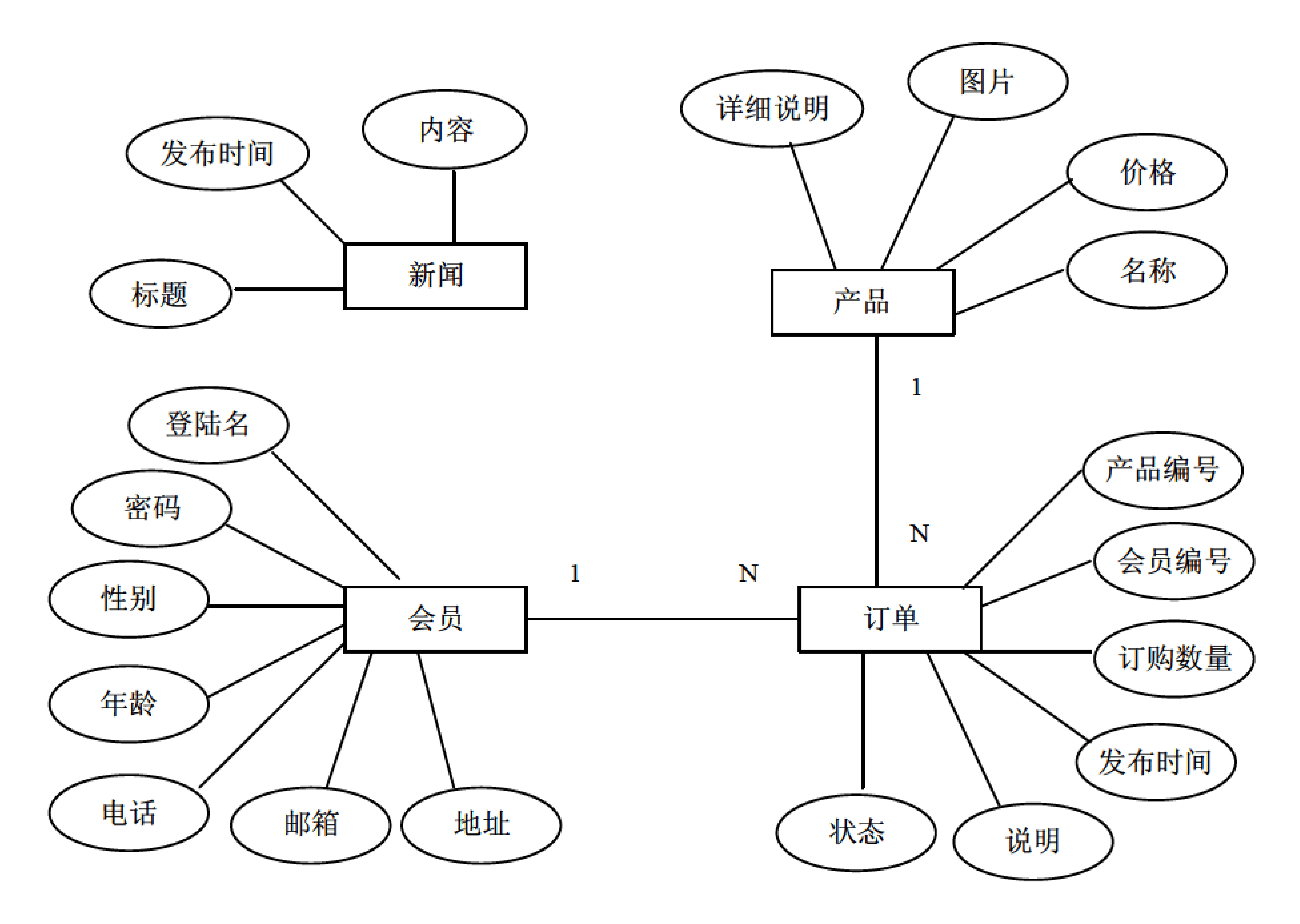


图 14-4　数据库实体关系(E-R)图

14.2.3　数据表设计

本系统采用 MS SQLServer 2005 数据库系统，首先建立一个名为 company_site 的数据库。这里数据库登录账号使用的是系统账号 sa，密码是 sa，下面是该数据库的结构信息。

1. 新闻信息表(tbl_news)

该表包含新闻 ID、标题、发布时间、新闻内容 4 个字段，如表 14-1 所示。

表 14-1　新闻信息表

列　名	类　型	是否为空	说　明
Id	Int	否	主键，自增列
n_title	nvarchar(100)	否	标题
n_publish_time	Datetime	否	发布时间
n_content	Text	否	新闻内容

2. 产品信息表(tbl_product)

该表包含产品 ID、产品名称、产品图片路径、产品价格、产品详细说明这 5 个字段，如表 14-2 所示。

表 14-2　产品信息表

列　名	类　型	是否为空	说　明
id	int	否	主键，自增列
p_name	nvarchar(50)	否	名称
p_picture	varchar(100)	否	图片路径
p_price	int	否	价格
p_details	text	否	详细说明

3．会员信息表(tbl_member)

包含会员 ID、登录名、密码、性别、年龄、电话、邮箱、地址 8 个字段，如表 14-3 所示。

表 14-3　会员信息表

列　名	类　型	是否为空	说　明
id	int	否	主键，自增列
m_login_name	varchar(20)	否	登录名，唯一
m_password	varchar(20)	否	密码
m_is_male	bit	否	性别
m_age	int	否	年龄
m_phone	varchar(20)	否	电话
m_email	varchar(50)	否	邮箱
m_address	varchar(255)	否	地址

4．会员信息表(tbl_order)

该表包含会员 ID、产品编号、会员编号、订购数量、发布时间、详细说明、订单状态这 7 个字段，订单状态为 false 时表示“未处理”，为 true 时表示“已处理”，如表 14-4 所示。

表 14-4　订单信息表

列　名	类　型	是否为空	说　明
id	int	否	主键，自增列
o_product_id	int	否	外建，tbl_product(id)，产品编号
o_member_id	int	否	外键，tbl_member(id)，会员编号
O_number	int	否	订购数量
O_publish_time	datetime	否	发布时间
o_details	text	否	详细说明
o_status	int	否	状态(是否处理)

14.2.4　数据表之间的关系

数据表之间的关系如图 14-5 所示。

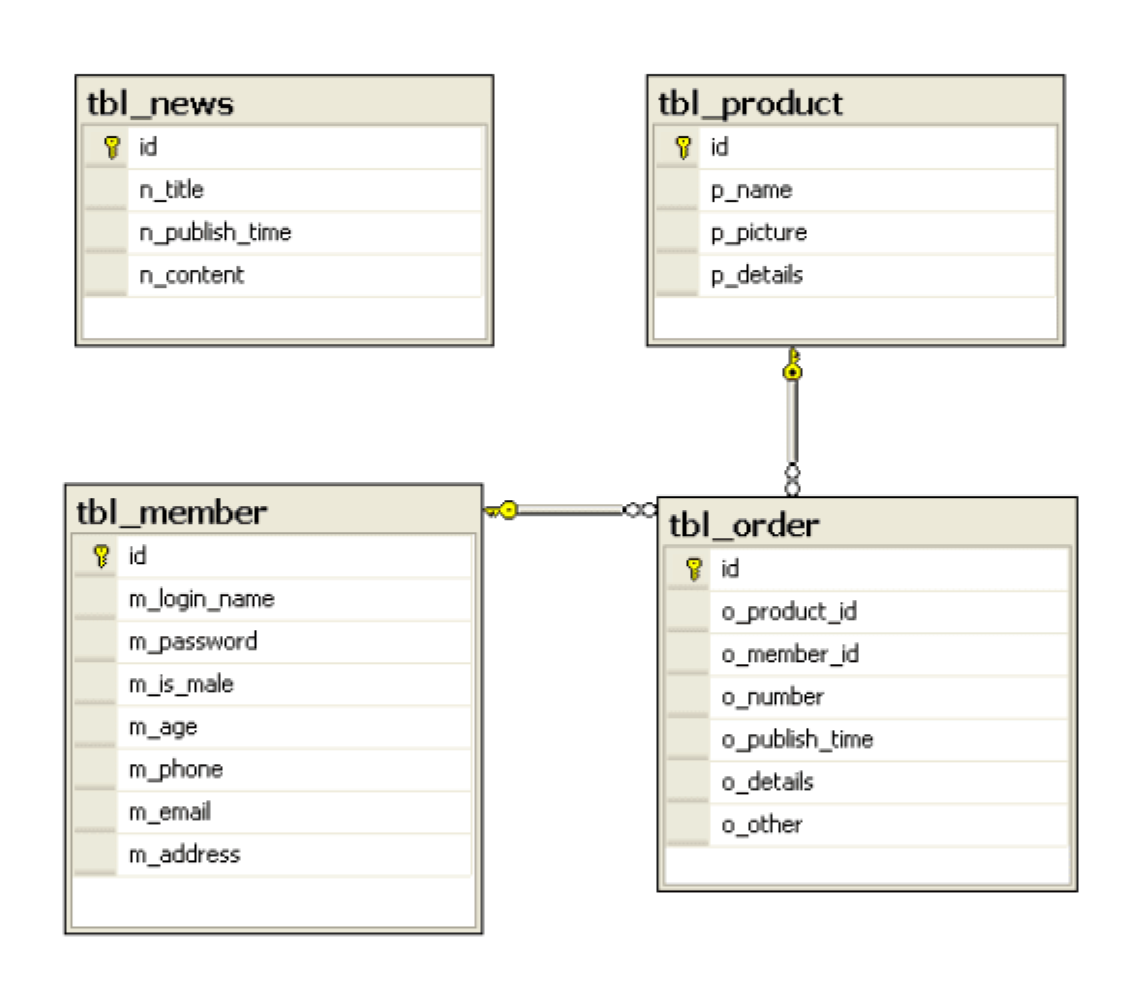


图 14-5　数据表之间的关系

14.3　公用模块编写

为了系统更好地移植和维护，本系统将最费时又没技术含量的部分抽取为系统公用模块，可以在程序里重复使用，大大加快了开发效率。

这一节介绍一下数据库连接和数据层的通用类 DBHelper。

14.3.1　编写数据库连接

为了方便应用程序移植，可以在应用程序配置文件 Web.Config 中设置数据库连接信息，将<connectionStrings />节点展开，添加一个链接字符串配置信息，如下所示：

```
<connectionStrings>
<add name="Sql2005" connectionString="Data Source=.;Initial
  Catalog=company_site;User ID=sa;Password=sa"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```



这里的 name 属性必需是 Sql2005，与后面将要提到的 DBHelper 类里的取值操作对应。这里使用的是 SQL Server 身份验证模式，你可以根据需要自行修改为 Windows 验证模式。

14.3.2 数据层类

为了方便操作数据库，将直接操作数据库的所有代码封装到 DBHelper 类里。DBHelper 类里首先要初始化一个数据库链接字符串：

```
private static readonly string _providerName = ConfigurationManager.
  ConnectionStrings["Sql2005"].ProviderName;    //用于创建数据提供者工厂
private static readonly string _connectionString =
  ConfigurationManager.ConnectionStrings["Sql2005"].ConnectionString;
```



这里的 Sql2005，和上一节我们讲到的链接字符串配置信息的 name 属性值对应。在这里还需要引用命名空间 System.Configuration 来操作配置文件。

下面是创建数据库连接的 GetConnection()方法：

```
private static DbConnection GetConnection()
{
    //创建数据提供者工厂
    DbProviderFactory factory =
      DbProviderFactories.GetFactory(_providerName);
    //用工厂创建数据库链接
    DbConnection connection = factory.CreateConnection();
    //设置链接字符串
    connection.ConnectionString = _connectionString;
    return connection;
}
```

下面是创建 Command 对象的 GetCommand()方法：

```
private static DbCommand GetCommand(string commandText,
  DbConnection connection)
{
    //创建命令
    DbCommand command = connection.CreateCommand();
    //设置命令字符串
    command.CommandText = commandText;
    //设置 DbCommand 对象的类型
    command.CommandType = CommandType.Text;
    return command;
}
```

下面是封装 Parameter 对象的 GetParameter()方法：

```
private static DbParameter GetParameter(string paramName, object paramValue,
  DbCommand command)
{
    //创建一个 DbCommand 对象的参数
    DbParameter parameter = command.CreateParameter();
    //设置参数名称
```

```
    parameter.ParameterName = paramName;
    //设置参数值
    parameter.Value = paramValue;
    return parameter;
}
```

下面是封装命令参数列表的 PackingParameters()方法：

```
private static void PackingParameters(Dictionary<string, object> parms,
  DbCommand command)
{
    if (parms != null)
    {
        //遍历参数字典
        foreach (var p in parms)
        {
            command.Parameters.Add(GetParameter(p.Key, p.Value, command));
        }
    }
}
```

下面是 ExecuteNonQuery()方法，功能是执行一个带参数列表的无返回值的 SQL 语句：

```
public static void ExecuteNonQuery(string sqlCommand,
  Dictionary<string, object> parms)
{
    //用 using 语句监控 DbConnection 对象，以保证资源释放
    using (DbConnection connection = GetConnection())
    {
        //创建 DbCommand 对象
        DbCommand command = GetCommand(sqlCommand, connection);
        PackingParameters(parms, command);  //封装参数列表
        connection.Open();  //打开数据库链接
        command.ExecuteNonQuery();  //执行 DbCommand 命令
        command.Parameters.Clear(); //清理参数
    }
}
```

下面是 ExecuteReader()方法，功能是执行一个带参数列表的返回一个结果集的 SQL 语句：

```
public static DbDataReader ExecuteReader(string sqlCommand,
  Dictionary<string, object> parms)
{
    //创建 DbConnection 对象，这里不能用 using，因为要返回 DataReader，
    //而 connection 一断开，DataReader 就无效了
    DbConnection connection = GetConnection();
    //创建 DbCommand 对象
    DbCommand command = GetCommand(sqlCommand, connection);
    PackingParameters(parms, command);  //封装参数列表
    connection.Open(); //打开数据库链接
    //调用 ExecuteReader 时，传递 CommandBehavior.CloseConnection 参数，
```

```
    //这样 reader 一关闭，连接也同时关闭
    DbDataReader reader =
      command.ExecuteReader(CommandBehavior.CloseConnection);
    command.Parameters.Clear(); //清理参数
    return reader;
}
```

下面是 ExecuteScalar()方法，功能是执行一个带参数列表的返回单个 int 型值的 SQL 语句，常用于统计数据：

```
public static int ExecuteScalar(string sqlCommand,
  Dictionary<string, object> parms)
{
    int count = 0;
    //用 using 语句监控 DbConnection 对象，以保证资源释放
    using (DbConnection conn = GetConnection())
    {
        //创建 DbCommand 对象
        DbCommand command = GetCommand(sqlCommand, conn);
        PackingParameters(parms, command); //封装参数列表
        count = (int)command.ExecuteScalar();  //执行命令
    }
    return count;
}
```

14.4 后台登录页面

登录页面比较简单，但是在本实例数据安全上起着至关重要的作用。下面我们来看一下它的实现步骤。

首先需要页面展示，这里用了两个文本框来获取用户输入的用户名和密码，用了一个图片按钮接收用户命令。主要代码如下：

```
<asp:TextBox ID="txtLoginName" runat="server" MaxLength="30"
  CssClass="input"></asp:TextBox>
<asp:TextBox ID="txtPassword" runat="server" MaxLength="30" CssClass="input"
  TextMode="Password"></asp:TextBox>
<asp:ImageButton ID="imgbtnLogin" runat="server"
  ImageUrl="images/bt_login.gif" Height="18" Width="70"
  onclick="imgbtnLogin_Click" />
```

在这里，接收密码的文本框和 TextMode 属性设置为 Password，这样用户输入的时候页面上显示的就不是真实的密码了。

命令按钮 imgbtnLogin 接收 onclick 事件，在用户单击该按钮时将触发 imgbtnLogin_Click 事件。我们来看一下事件处理程序代码：

```
protected void imgbtnLogin_Click(object sender, ImageClickEventArgs e)
```

```
{
    string username = this.txtLoginName.Text.Trim();
    string password = this.txtPassword.Text;
    if (username=="admin" && password=="admin")  //验证用户
    {
        Session["CurrentAdmin"] = "admin";  //记录用户验证状态
        Response.Redirect("Main.aspx");  //跳转到管理主页面
        return;
    }
    Response.Write("<script>alert('用户名或密码错误！\\n 请重新登录。');
                    location='Default.aspx';</script>");
    Response.End();
}
```

这里首先 new 了两个对象，存储用户输入的用户名和密码，然后验证用户名和密码。如果验证成功，记录用户登录状态，跳转到管理主页面，如果验证不成功，提示并让用户重新登录。



验证用户的过程本该从数据库中获取信息验证，这里为了方便，直接用指定的账号密码验证。

登录页面为了美观，也通常找专业的美工设计或到网上找模版套用。如图 14-6 所示是登录页面的运行效果。



图 14-6　登录页面的运行效果

14.5　管理员界面：新闻管理

通过了权限认证，我们就进入了神秘的后台管理界面。

系统第一个模块非常简单：新闻管理，这里只要实现添加和删除功能即可。

14.5.1 新闻的添加

新闻有 3 个属性：标题、发布时间和新闻内容。发布时间当然就是新闻提交的时间了，这里由系统自动获取，所以用户只要填入标题和新闻内容就可以了。

新闻添加的界面十分简洁，如图 14-7 所示。

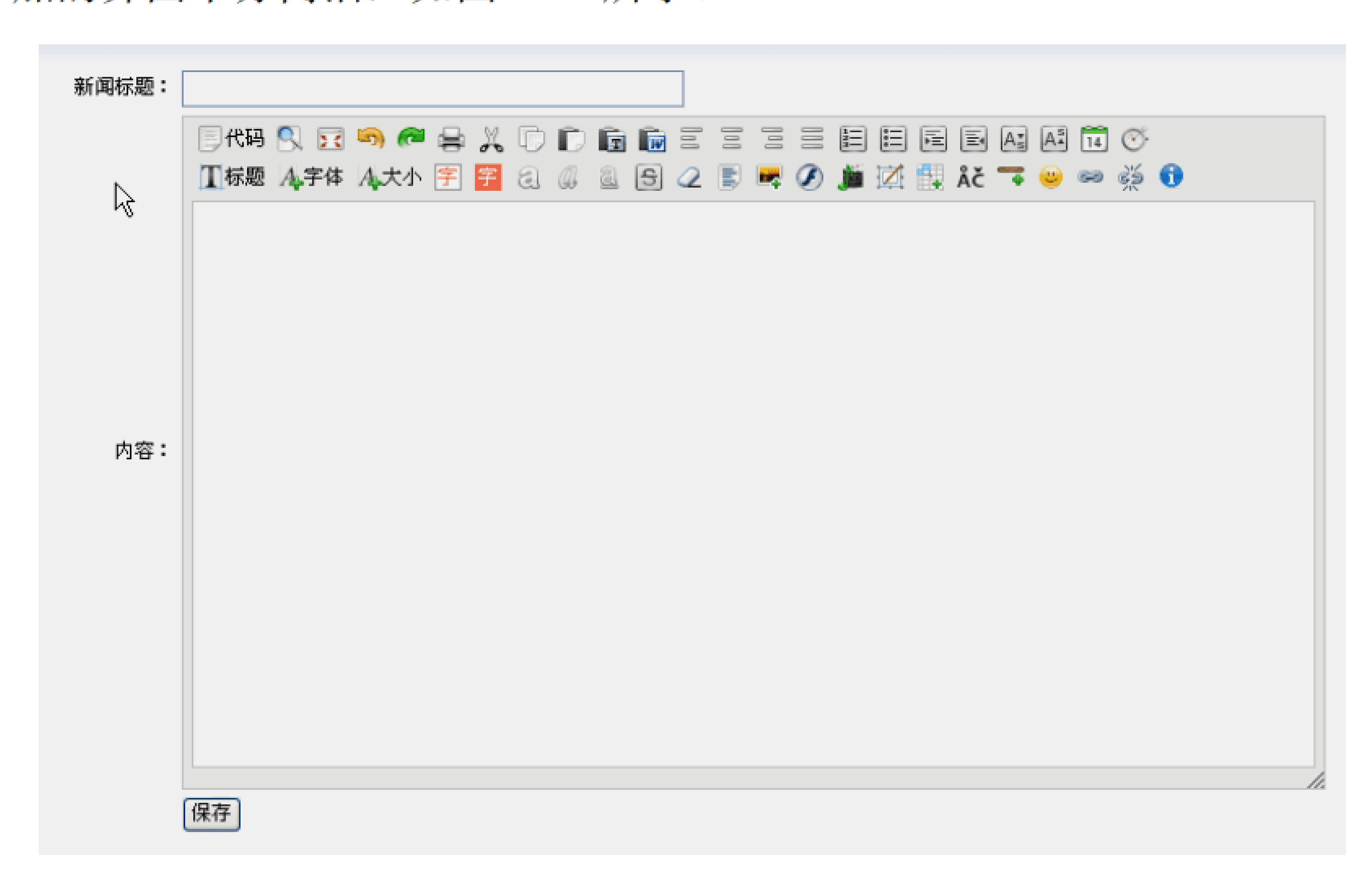


图 14-7 添加新闻页面

界面很简洁吧！

不过似乎读者可能觉得有点不太对劲儿，那个有宽宽的灰边的，上面还有两行类似 Word 工具条一样的玩意儿是什么东西呢？找遍了整个 VS 工具箱都没有找到。

说实话，作者也没找到……

我们平时看新闻时会注意到，有些文章里会有类似 Word 编辑出来一样的效果。比如加粗，文字颜色、字体、字号、插入图片、表格、Flash……之类的效果，是怎么实现的呢？用 Dreamweaver？绝对没问题，但不会有人那么白痴地一个个页面用 Dreamweaver 处理的。所以要解决这个问题，就要引入功能强大的“JavaScript 文本编辑器”。

这种“JavaScript 文本编辑器”有很多，百度一下，一排排地列出来任你挑。这里作者用的是 KindEditor 3.3.1，是从网盘里淘出来的，版本有点老，不过够用了。

使用起来非常简单，将包含 kindeditor.js 文件的文件夹拷入 Web 项目目录。这里放在与新闻添加页面同级的目录下了。页面头部加入以下代码就可以了：

```
<script src="kindeditor-3.3.1/kindeditor.js" charset="utf-8"
  type="text/javascript"></script>
<script type="text/javascript" language="javascript">
    KE.show({
    id: '<%= txtContent.ClientID %>',❶
    cssPath : 'kindeditor-3.3.1/examples/index.css'
});
</script>
```

下面是正文内容的服务器控件代码：

```
<asp:TextBox ID="txtContent" runat="server"
  style="width:100%;height:400px;visibility:hidden;"></asp:TextBox>
```

上面 Javascript 代码里❶这一行单引号里写的应该是页面 HTML 文本框的 id，但是这里用的是服务器控件，就需要这样写以便动态地指定。



运行的时候可能会报一个 HttpRequestValidationException 异常，不过不要紧，是因为你提交的信息里包含了 HTML 标签，ASP.NET 认为这可能是不安全的，解决办法是在页面 Page 标签里设置 validateRequest 值为 false 即可。

“标题”文本框和“保存”按钮的代码就不贴了。下面来看看“保存”按钮的单击事件处理程序：

```
protected void btnSave Click(object sender, EventArgs e)
{
    NewsInfo news = this.PackagingNewsForPage(); //封装新闻信息
    NewsManager.PublishNews(news);  //发布新闻
    Response.Write("<script>alert('发布成功。点确定返回新闻列表。');
                     location='NewsList.aspx'; </script>");
}
private NewsInfo PackagingNewsForPage() // 封装新闻信息
{
    NewsInfo news = new NewsInfo();
    news.PublishTime = DateTime.Now;
    news.Title = this.txtTitle.Text;
    news.Content = this.txtContent.Text;
    return news;
}
```

至于业务层，代码就不贴了，无非就是代理一下数据库操作，没有什么业务逻辑。

下面是数据访问层：

```
public void Append(NewsInfo news)
{
    //创建 SQL 语句
    string sql = "insert into tbl_news values(@Title, @PublishTime, @Content)";
    //创建字典对象封装参数
    Dictionary<string, object> prams = new Dictionary<string, object>();
    prams.Add("@Title", news.Title);
    prams.Add("@PublishTime", news.PublishTime);
    prams.Add("@Content", news.Content);
    DBHelper.ExecuteNonQuery(sql, prams);  //执行带参数的 SQL 语句
}
```

代码完成了。

运行程序，输入文字，编辑样式，插入图片……确定保存。

怎么样？一则图文并茂的新闻就发布成功了。

在浏览器中查看，效果如图 14-8 所示。



图 14-8　新闻查看页面

14.5.2　新闻的删除

要删除新闻，就得先找到它。最好的方式是把所有新闻列出来，选择想要删除的新闻，执行删除操作。所以整个过程执行两个操作：列表、删除。

列表要尽可能多地列出重要信息，以便精确地确认某条记录。这里由于页面篇幅问题只列出标题和发布时间，核心代码如下：

```
<asp:Repeater ID="rptNews" runat="server">
<ItemTemplate>
<tr>
<td><%# Eval("Title") %></td>
<td><%# Eval("PublishTime") %></td>
<td>
<asp:LinkButton runat="server" ID="lnkbtnDeletes"
  OnClientClick="return confirm('确定删除？');"
  CommandArgument='<%# Eval("ID") %>'
  OnCommand="lnkbtnDelete_Command">删除
</asp:LinkButton>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
```

以上代码使用 Repeater 遍历并重复显示数据源里的内容。

访问数据库并绑定 Repeater 的数据源为返回的集合，即可完成对 Repeater 的填充：

```
protected void BindList()
{
```

```
    this.rptNews.DataSource = NewsManager.GetNewsList();
    this.rptNews.DataBind();
}
```

事件处理程序 lnkbtnDelete_Command 根据绑定在 LinkButtion 对象的 CommandArgument 属性的 ID 值区分每一条新闻，并完成对该条新闻的删除操作。具体代码如下：

```
protected void lnkbtnDelete Command(object sender, CommandEventArgs e)
{
    int id = int.Parse(e.CommandArgument.ToString());
    NewsManager.DeleteNewsById(id); //根据 ID 删除新闻
    this.BindList(); //删除后重新绑定列表，使页面与数据库同步
}
```

执行完删除操作，别忘了再对 Repeater 的数据源绑定一次，因为数据库已经变了，但页面上的内容还没有改变，再绑定一次，保证数据库和页面上的数据统一。

别看页面 Repeater 列出所有记录信息的时候那么简单，数据库操作可不是那么容易了：

```
public IList<NewsInfo> GetNewss()
{
    string sql = "select * from tbl_news order by id desc"; //创建 SQL 语句
    return this.GetNewsBySql(sql);
}
private IList<NewsInfo> GetNewsBySql(string sql)
{
    IList<NewsInfo> newsList = new List<NewsInfo>();
    //用 using 语句管理 reader，使用完自动释放资源
    using (DbDataReader reader = DBHelper.ExecuteReader(sql))
    {
        //循环遍历 reader，取出值并封装到集合中
        while (reader.Read())
        {
            newsList.Add(this.GetNewsByReader(reader));
        }
    }
    return newsList;
}
private NewsInfo GetNewsByReader(DbDataReader reader)
{
    /*
    * 从 reader 中取值并封装 News 对象
    */
    NewsInfo newNews = new NewsInfo();
    newNews.ID = reader.GetInt32(0);
    newNews.Title = reader.GetString(1);
    newNews.PublishTime = reader.GetDateTime(2);
    newNews.Content = reader.GetString(3);
    return newNews;
}
```



为了代码重用，这里较多地使用了封装。把一段代码封装成方法，执行其他功能时可以有效减少代码量。比如这个实例中一直使用的 DBHelper 类。

相对于查询来说，删除的代码就简单了不少：

```
string sql = "delete tbl_news where id = " + id;
DBHelper.ExecuteNonQuery(sql);
```

两行代码就执行了删除操作，看起来“封装”的魔力还是挺强大的嘛！

14.6 管理员界面：产品管理

产品管理相对于新闻来说稍微复杂一点，需要上传保存图片，还需要修改功能，修改的时候图片的处理也是一个挺让人郁闷的环节。

不过再大的郁闷也挡不住我们探究学习的步伐。不要被我的夸张吓倒，当做出来以后，就会觉得根本就是很简单的事情。

14.6.1 产品的添加

产品对象有以下属性：名称、图片、价格、详细说明。

来看服务器端控件代码：

```
<asp:TextBox ID="txtName" runat="server" Width="300" MaxLength="50">
</asp:TextBox>
<asp:FileUpload ID="fuPicture" runat="server" Width="300" />
<asp:TextBox ID="txtPrice" runat="server" Width="300"></asp:TextBox>
<asp:TextBox ID="txtDetails" runat="server"
  style="width:600px;height:230px;">
</asp:TextBox>
<asp:Button runat="server" ID="btnSave" Text="保存" onclick="btnSave_Click" />
```

再看“保存”按钮的 onclick 事件处理程序：

```
static string uploadFolder = "upload/";
protected void btnSave_Click(object sender, EventArgs e)
{
    //如果没有上传文件，或上传的文件不是 JPG 格式的，提示并返回
    if (!fuPicture.HasFile || !this.CheckFileExtendsName(fuPicture.FileName))
    {
        divResult.InnerHtml = "<script>alert('请上传 JPG 格式的图片!');</script>";
        return;
    }
    //获取页面上的数据
    ProductInfo product = this.PackagingProductForPage();
    //保存文件到指定位置
    fuPicture.SaveAs(Server.MapPath("~/" + product.Picture));
```

```
    //将产品信息保存到数据库
    ProductManager.AppendProduct(product);
    divResult.InnerHtml = "<script>alert('上传成功。单击确定返回产品管理列表。');
                            location='ProductList.aspx';</script>";
}
private bool CheckFileExtendsName(string filename)
{
    //用正则表达式验证文件名的扩展名是不是 jpg 或者 jpeg
    Regex picExtendsName = new Regex("(.jpg|.jpeg)$");
    return picExtendsName.IsMatch(filename.ToLower());
}
private ProductInfo PackagingProductForPage()
{
    ProductInfo product = new ProductInfo();
    product.Details = this.txtDetails.Text;
    product.Name = this.txtName.Text;
    try { product.Price = int.Parse(this.txtPrice.Text); }
    catch { }
    //创建一个图片路径，规则是
    //上传文件夹地址+当前年份+当前月份+当前日+(1000000—9999999)的一个数字+“.jpg”
    product.Picture = uploadFolder
        + DateTime.Now.Year + DateTime.Now.Month
        + DateTime.Now.Day + new Random().Next(1000000, 9999999) + ".jpg";
    return product;
}
```

数据访问层代码和新闻添加的时候差不多，无非是创建 SQL 语句，绑定参数，执行一下就完毕，这里就不贴代码浪费篇幅了。

很简单吧！

14.6.2　产品的删除和更新

执行产品的更新和删除操作都得先找出来，也就是先列出来，找到要更新或删除的项，执行相应的操作。

同样，这里的列表功能也是用 Repeater 控件实现：

```
<asp:Repeater ID="rptProducts" runat="server">
<ItemTemplate>
<tr>
<td><%# Eval("Name") %></td>
<td><%# Eval("Price") %></td>
<td>
<asp:LinkButton runat="server" ID="lnkbtnUpdates"
  PostBackUrl='<%# "EditProduct.aspx?id=" + Eval("ID") %>'>更新
</asp:LinkButton>
</td>
<td>
```

```
<asp:LinkButton runat="server" ID="lnkbtnDeletes"
OnClientClick="return confirm('删除该产品会删除与其相关的所有订单。\n 确定删除？');"
CommandArgument='<%# Eval("ID") %>' OnCommand="lnkbtnDelete_Command">删除
</asp:LinkButton>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
```

同样，绑定是如此简单：

```
protected void BindList()
{
    this.rptProducts.DataSource = ProductManager.GetProductList();
    this.rptProducts.DataBind();
}
```

删除操作也是如此简单：

```
protected void lnkbtnDelete_Command(object sender, CommandEventArgs e)
{
    int id = int.Parse(e.CommandArgument.ToString());
    ProductManager.DeleteById(id);
    this.BindList(); //删除后重新绑定列表，使页面与数据库同步
}
```

但这里要注意了，因为产品表和订单表有主外键关系，所以直接删除产品时可能会触发异常，所以删除产品以前需要先删除引用该产品的订单，这里的业务逻辑层就需要简单的业务逻辑了：

```
public static void DeleteById(int id)
{
    //OS、PS 是数据访问层对象
    OS.DeleteByProductID(id);
    PS.DeleteById(id);
}
```

数据访问层和删除新闻的代码类似，只把 SQL 语句里的表名改一下就完了，不再多说。这里重点讲一下产品的更新操作。

产品更新的页面控件代码和添加时候一样，直接 Copy 就行了。

现在看 cs 文件里的主要代码：

```
static string uploadFolder = "upload/";
protected void Page_Load(object sender, EventArgs e)
{
    divResult.InnerHtml = "";
    if (!IsPostBack)
    {
        this.BindProduct();
    }
}
```

```
protected void btnSave_Click(object sender, EventArgs e)
{
    //封装产品对象
    ProductInfo product = this.PackagingProductForPage();
    //如果上传控件存在文件
    if (fuPicture.HasFile)
    {
        //检查文件扩展名
        if (!this.CheckFileExtendsName(fuPicture.FileName))
        {
            divResult.InnerHtml =
              "<script>alert('请上传 JPG 格式的图片！');</script>";
            return;
        }
        //保存新文件至硬盘
        fuPicture.SaveAs(Server.MapPath("~/" + product.Picture));
        //更新产品信息
        ProductManager.ModifyProduct(product);
    }
    else
    {
        //更新除图片之外的产品信息
        ProductManager.ModifyProductNoPicture(product);
    }
    divResult.InnerHtml = "<script>alert('修改成功。单击确定返回产品管理列表。');
                           location='ProductList.aspx';</script>";
}
private void BindProduct()
{
    try
    {
        int id = int.Parse(Request["id"]);
        ProductInfo product = ProductManager.GetProductById(id);
        this.txtDetails.Text = product.Details;
        this.txtName.Text = product.Name;
        this.txtPrice.Text = product.Price.ToString();
    }
    catch { }
}
private bool CheckFileExtendsName(string filename)
{
    //用正则表达式验证文件名的扩展名是不是 jpg 或者 jpeg
    Regex picExtendsName = new Regex("(.jpg|.jpeg)$");
    return picExtendsName.IsMatch(filename.ToLower());
}
private ProductInfo PackagingProductForPage()
{
    ProductInfo product = new ProductInfo();
    //产品 ID 就是请求过来时传递的参数 id
```

```
    try { product.ID = int.Parse(Request["id"]); }
    catch { }
    product.Details = this.txtDetails.Text;
    product.Name = this.txtName.Text;
    try { product.Price = int.Parse(this.txtPrice.Text); }
    catch { }
    //创建一个图片路径，规则是
    //上传文件夹地址+当前年份+当前月份+当前日+(1000000－9999999)的一个数字+“.jpg”
    product.Picture = uploadFolder
        + DateTime.Now.Year + DateTime.Now.Month
        + DateTime.Now.Day + new Random().Next(1000000, 9999999) + ".jpg";
    return product;
}
```

开始打开这个页面时，会传过来一个产品信息的 id，根据这个 id，查询数据库，填充到页面控件里。

在单击“保存”按钮时，先封装页面更改过的信息；如果用户上传的有文件，就测试上传的文件类型是不是 JPG，如果不是提示用户，如果是，保存该文件并更新数据库；如果用户上传的没有文件，就直接更新数据库信息，但不更新 Picture 属性。

业务逻辑层代码如下：

```
public static void ModifyProduct(ProductInfo product)
{
    //PS 是数据访问层对象
    PS.Update(product);
}

public static void ModifyProductNoPicture(ProductInfo product)
{
    //PS 是数据访问层对象
    ProductInfo oldProduct = PS.GetById(product.ID);
    product.Picture = oldProduct.Picture;
    PS.Update(product);
}
```

上面的两段代码中，第一个是直接更新，第二个是先从数据库读取原来的产品信息，并把原来的 Picture 属性更新到用户更改过的产品信息里，把用户更改过的产品信息更新至数据库。

至此大功告成。代码是稍微啰嗦了那么一点点，但更新功能还是完美地实现了，再看起来就容易多了。

14.7　管理员界面：会员管理

用户是企业的生命之源，数据库里存储的这些会员信息就是企业的无形资产，堪称无价。但这么重要的信息资源，当然需要我们去使用、去管理、去维护了。

14.7.1　会员信息查看

像这种无形资产，看不到它等于没有。所以，管理员能查看会员信息是最重要的事情。这里同样使用 Repeater 控件实现列表功能：

```
<asp:Repeater ID="rptMembers" runat="server">
<ItemTemplate>
<tr>
<td><%# Eval("LoginName") %></td>
<td><%# bool.Parse(Eval("IsMale").ToString())?"男":"女" %></td>
<td><%# Eval("Age") %></td>
<td><%# Eval("Phone") %></td>
<td>
<asp:LinkButton runat="server" ID="lnkbtnUpdates"
  PostBackUrl='<%# "ShowMember.aspx?id=" + Eval("ID") %>'>查看详细
</asp:LinkButton>
</td>
<td>
<asp:LinkButton runat="server" ID="lnkbtnDelete"
  OnClientClick="return confirm('是否删除。')"
  CommandArgument='<%# Eval("ID") %>'
  OnCommand="lnkbtnDelete_Command">删除该用户
</asp:LinkButton>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
```

这里只列出登录名、性别、年龄和电话 4 项信息。其他的在“查看详细”页面会一一列出：

```
private void BindMembers()
{
    this.rptMembers.DataSource = MemberManager.GetMemberList();
    this.rptMembers.DataBind();
}
```

同样，这是 Repeater 控件数据源绑定代码。

从上面的页面代码我们可以看到，“查看详细”链接跳转到了另一个新的页面来展示会员详细信息。该页面只是一系列的 Label 控件，用以绑定会员信息，这里不再列出页面代码，只贴出绑定方法代码如下：

```
private void BindMember()
{
    try
    {
```

```
        int id = int.Parse(Request["id"]);
        MemberInfo member = MemberManager.GetMemberById(id);
        this.lblAddress.Text = member.Address;
        this.lblAge.Text = member.Age.ToString();
        this.lblEmail.Text = member.Email;
        this.lblLoginName.Text = member.LoginName;
        this.lblPassword.Text = member.Password;
        this.lblPhone.Text = member.Phone;
        this.lblSex.Text = member.IsMale ? "男" : "女";
    }
    catch { }
}
```

至此就完成了会员信息查看功能。由于比较简单，在这里就不过多说明了。

14.7.2 会员信息删除

有时候会有一些访客注册一些不完整信息的账号，或恶意注册的账号一直存放在数据库里浪费空间而且影响查看，所以应提供删除功能，使管理员能方便地删掉毫无价值的会员账号。

细心的读者或许已经发现，“会员信息查看”实例中的页面代码中已经包含删除会员功能操作按钮，所以这里只展示事件处理程序代码就可以了：

```
protected void lnkbtnDelete_Command(object sender, CommandEventArgs e)
{
    int id = int.Parse(e.CommandArgument.ToString());
    MemberManager.DeleteById(id);
    this.BindMembers();
}
```

这里也要注意一下业务逻辑层代码：

```
public static void DeleteById(int id)
{
    //OS 是数据访问层对象
    OS.DeleteByMemberID(id);
    MS.DeleteById(id);
}
```

同样因为会员表和订单表主外键关系，删除会员需要先删除隶属该会员的订单信息。

数据访问层代码参照新闻管理或产品管理的代码，这里不过多解释。

14.8 管理员界面：订单管理

本系统也小小地实现了一下电子商务的功能，也就是现在要讲的订单管理。在线订单功能无疑节省了大量的人力资源和时间资源。

14.8.1　订单的查看

同样用 Repeater 展示出所有订单，下面是主要代码：

```
<asp:Repeater ID="rptOrders" runat="server">
<ItemTemplate>
<tr>
<td>产品名称：<%# ConvertProduct(Eval("ProductID")) %></td>
<td>会员 ID：<%# ConvertMember(Eval("MemberID")) %></td>
<td>订购数量：<%# Eval("Number") %></td>
<td>发布时间：<%# Eval("PublishTime") %></td>
</tr>
<tr>
<td colspan="5">
详细说明：<%# Eval("Details") %><br /><br />
<div style=" text-align:right;">
<asp:LinkButton runat="server" ID="lnkbtnChangeStatus"
  CommandArgument='<%# Eval("ID") %>'
  OnCommand="lnkbtnChangeStatus Command"
  Visible='<%# !bool.Parse(Eval("Status")+"") %>'>处理订单
</asp:LinkButton>
<span <%# !bool.Parse(Eval("Status")+"")?"style='display:none'":"" %> >已处理
</span>
</div>
</td>
</tr>
</ItemTemplate>
</asp:Repeater>
```

订单内容不多，这里直接在列表显示所有信息。

Repeater 数据源绑定代码如下：

```
private void BindList()
{
    this.rptOrders.DataSource = OrderManager.GetOrderList();
    this.rptOrders.DataBind();
}
```

这里要注意的是 ConvertProduct(object)方法和 ConvertMember(object)方法，这两个方法是将对应的 ID 转换为对应的名称字符，源代码如下：

```
protected string ConvertProduct(object oid)
{
    try
    {
        int id = int.Parse(oid.ToString());
        return ProductManager.GetProductById(id).Name;
    }
    catch { return ""; }
```

```
}
protected string ConvertMember(object oid)
{
    try
    {
        int id = int.Parse(oid.ToString());
        return MemberManager.GetMemberById(id).LoginName;
    }
    catch { return ""; }
}
```



这里着重说明一下，这两个方法的作用域必需大于或等于 protected，否则前台页面访问不到；而且方法参数只能是 object 类型，因为数据绑定方法 Eval 或 Bind 返回的都是 object 对象。

14.8.2 订单的处理

这里订单的处理比较简单，只是将订单状态更改一下就行了。继前面给出的页面代码之后，lnkbtnChangeStatus 控件的 OnCommand 事件处理程序如下：

```
protected void lnkbtnChangeStatus_Command(object sender, CommandEventArgs e)
{
    int id = int.Parse(e.CommandArgument.ToString());
    OrderManager.ChangeStatus(id);
    this.BindList(); //删除后重新绑定列表，使页面与数据库同步
}
```

代码只调用业务逻辑层的 ChangeStatus(int)方法，实际上业务逻辑层根据 ID 读出该对象，更改状态后更新至数据库。业务逻辑层代码如下：

```
public static void ChangeStatus(int id)
{
    //OS 是数据访问层对象
    OrderInfo order = OS.GetByID(id);
    order.Status = !order.Status;
    OS.Update(order);
}
```



详细的数据访问层代码这里不再贴出，读者请参考前几节的内容。

14.9 用 户 界 面

界面对程序员来说不重要，美观程度依各个网站美工水平而定。这里我们只需要到网上找

现成的页面，把程序填进去就可以了。作者个人比较喜欢简洁，所以选的页面模版也相对来说比较简单。

14.9.1　网站首页

首页整体上比较简洁，我们先看效果，如图 14-9 所示。



图 14-9　企业网站首页的效果

页面左边是一个导航，右边内容区里有一个新闻列表和产品列表，为了页面美观，新闻只显示最新的 5 条，产品只显示最新上传的 6 个。

新闻只需要列表显示就完了，按照习惯，还是用 Repeater 控件；产品为了美观，需要对列表布局稍做要求，这里显示两行，一行 3 个，不过 ASP.NET 也提供了现成的数据控件：DataList，它能自动生成表格，根据开发者的设置，自动生成 N 列的表格，一个单元格就是一条数据。下面是页面的代码：

```
<h2><a href="#">公司新闻</a></h2>
<ul>
   <asp:Repeater ID="rptNews" runat="server">
   <ItemTemplate>
   <li>
      <a href='<%# "News.aspx?id=" + Eval("ID") %>'>
         <%# Eval("Title") %></a>
      [<%# Eval("PublishTime") %>]
   </li>
   </ItemTemplate>
   </asp:Repeater>
</ul>
<h2><a href="#">创意产品</a></h2>
```

```
<p>
   <asp:DataList ID="dlProduct" runat="server" RepeatColumns="3">
   <ItemTemplate>
   <a href='<%# "Product.aspx?id=" + Eval("ID") %>'>
      <img alt='<%# Eval("Name") %>' width="150" height="180"
         src='<%# Eval("Picture") %>'
         style=" margin:5px; border:solid 1px #666;" />
   </a>
   </ItemTemplate>
   </asp:DataList>
</p>
```

从上面代码中我们可以发现，DataList 在这里多设置了一个 RepeatColumns 属性值为 3，意思就是该数据列表有 3 列：

```
private static int NEWS_LIST_SIZE = 5; //新闻列表大小
private static int PRODUCT_LIST_SIZE = 6;  //产品列表大小
protected void Page_Load(object sender, EventArgs e)
{
   this.BindNews();
   this.BindProduct();
}
private void BindNews()
{
   this.rptNews.DataSource = NewsManager.GetNewsList(NEWS_LIST_SIZE);
   this.rptNews.DataBind();
}
private void BindProduct()
{
   this.dlProduct.DataSource =
     ProductManager.GetProductList(PRODUCT_LIST_SIZE);
   this.dlProduct.DataBind();
}
```

在这段代码中，我们发现调用业务层方法绑定列表值时多加了一个参数，这里表示是要获取起始的 N 条记录。

我们来看业务层代码就明白了：

```
public static IList<NewsInfo> GetNewsList(int topNumber)
{
   return NS.GetNewss(0, topNumber);
}
```

传递过来的参数是最顶部的条数。当然，调用业务层方法又变了个样，需要两个参数：

```
public IList<NewsInfo> GetNewss(int startNumber, int length)
{
   string sql = "select top " + length
     + " * from tbl_news where id not in (select top "
     + startNumber + " id from tbl_news order by id desc) order by id desc";
```

```
    return this.GetNewsBySql(sql); ❶
}
```

代码很简单。看完后我们发现，两个参数一个是起始位置，一个是返回列表长度。至于这里调用的❶这个方法，前面贴过代码，这里不再多说。

这是新闻的列表，产品的列表基本一模一样，对照着参考一下就可以了。

14.9.2　新闻查看

新闻的列表页面代码可以直接复制网站首页的，只在绑定方法处略加修改即可：

```
private void BindNews()
{
    this.rptNews.DataSource = NewsManager.GetNewsList();
    this.rptNews.DataBind();
}
```

在新闻内容展示页面，需要向访客展示新闻的所有内容、标题、发布时间，以及图文并茂的内容。说到这里，提一下前面我们讲的“文本编辑控件”的原理：它也是根据用户的操作生成的 HTML 代码，加的行内样式表。这里只要将它摆到页面上就行了，所以，同样用一个 Label 标签足矣。

来看看展示页面的代码：

```
<h2>公司新闻</h2>
<center>
<h3><asp:Label ID="lblTitle" runat="server"></asp:Label></h3>
</center>
<p><asp:Label ID="lblContent" runat="server"></asp:Label></p>
<p style="text-align:right;">
<asp:Label ID="lblPublishTime" runat="server"></asp:Label>
</p>
```

如此简单，只需要 3 个 Label 就行了。来绑定一下数据：

```
private void BindNews()
{
    int id = 0;
    try { id = int.Parse(Request["id"]); }
    catch { }
    NewsInfo news = NewsManager.GetNewsById(id); //根据 ID 获得新闻
    //如果该新闻不存在，跳回新闻列表页面
    if (news == null) Response.Redirect("NewsList.aspx");
    this.lblContent.Text = news.Content;
    this.lblPublishTime.Text = news.PublishTime + "";
    this.lblTitle.Text = news.Title;
}
```

这里加了点数据安全验证，读者可以参考。不多说，先看运行效果，如图 14-10 所示。



图 14-10　新闻内容页面

怎么样？一个美轮美奂的新闻展示页面诞生了。读者还可以根据自己的需要，插入表格、插入 Flash、插入视频……

14.9.3　产品展示

同样，这里的产品列表页的内容也是从网站首页复制的，是不是有想把它做成用户控件的冲动呢？当然你可以自己去实现的。

数据绑定也是稍做修改，思路参考上节讲的新闻列表，这里不多说了。

产品展示页面要一个效果——简洁明了。这是源代码：

```
<h2>创意产品</h2>
<center>
<h3><asp:Label ID="lblName" runat="server"></asp:Label></h3>
</center>
<center><asp:Image ID="imgPicture" runat="server" Width="350" /></center>
<p><center>价格：<asp:Label ID="lblPrice" runat="server"></asp:Label>元  [
<asp:HyperLink ID="hlProduct" runat="server">订购该产品
</asp:HyperLink>]</center></p>
<p><asp:Label ID="lblDetails" runat="server"></asp:Label></p>
```

在这里我们展示出产品的名称、图片、价格、详细说明。当然，还可以直接订购。

看数据绑定代码：

```
private void BindProduct()
{
    int id = 0;
    try { id = int.Parse(Request["id"]); }
    catch { }
```

```
    ProductInfo product = ProductManager.GetProductById(id); //根据 ID 获得产品
    //如果该产品不存在，跳回产品列表页面
    if (product == null) Response.Redirect("ProductList.aspx");
    this.lblDetails.Text = product.Details;
    this.lblName.Text = product.Name;
    this.imgPicture.ImageUrl = product.Picture;
    this.lblPrice.Text = product.Price.ToString();
    this.hlProduct.NavigateUrl = "NewOrder.aspx?id=" + product.ID;
}
```

同样，我们做了简单的数据安全性验证，毕竟 GET 方式传参嘛，用户很可能在 URL 上做点手脚(也算是逗你玩儿啦)。所以我们验证一下，让程序更安全、更健壮。

执行一下，看看效果，如图 14-11 所示。



图 14-11　产品详细说明

看到我们如此轻松地完成了任务，很是欣慰！

14.9.4　会员信息管理

会员信息管理是积累客户量、吸引新客户、留住老客户的有效方法。在这里我们的会员可以注册账号，修改会员信息。当然还有最重要的操作，即订购产品。这里我们先介绍会员的注册、登录和管理个人信息。

在未登录的情况下点击导航栏的“会员中心”或者产品详情页的“订购该产品”链接，我们就会进入用户登录页面。如果有账号，可以直接登录，如果没有，在这里单击“注册”按钮，就可以进入会员注册页面了，如图 14-12 所示。

在这里，填入想好的用户名、密码和其他详细信息，单击“提交”按钮，就可以提交信息。当然，如果用户名存在的话，是不可能注册成功的。密码必填，而且两次输入必须一致。其他

项没过多要求，但建议填真实信息，以便日后客服联系。

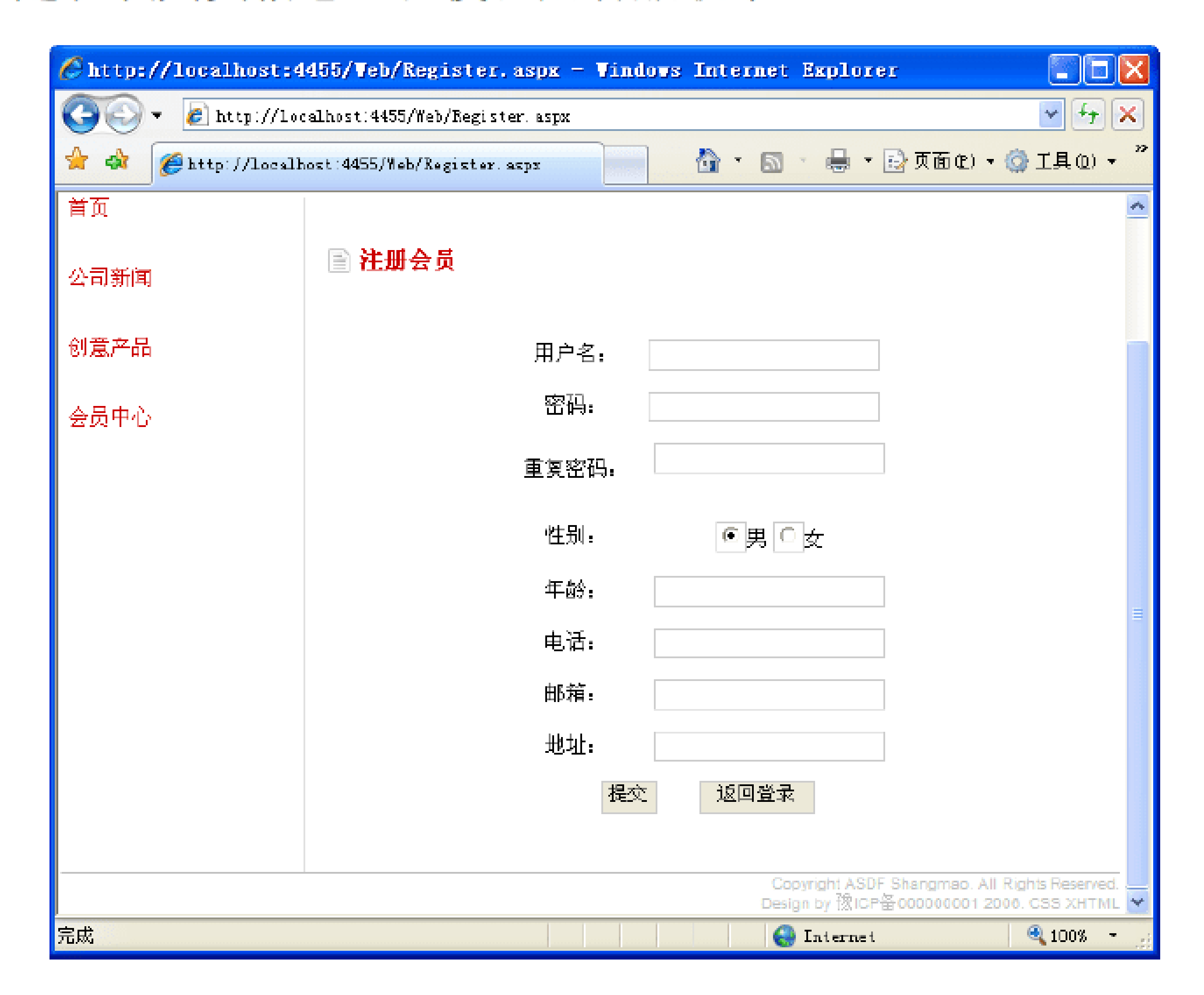


图 14-12　会员注册页面

下面是页面的主要代码：

```
<table cellpadding="5">
<tr>
<td>用户名：</td>
<td>
<asp:TextBox ID="txtLoginName" runat="server" MaxLength="20"></asp:TextBox>
<asp:RequiredFieldValidator ID="rfvUsername" runat="server"
  ControlToValidate="txtLoginName" ErrorMessage="*">
</asp:RequiredFieldValidator>
</td>
</tr>
<tr>
<td>密码：</td>
<td>
<asp:TextBox ID="txtPassword" runat="server"
  TextMode="Password" MaxLength="20">
</asp:TextBox>
<asp:RequiredFieldValidator ID="rfvPassword" runat="server"
  ControlToValidate="txtPassword" ErrorMessage="*">
</asp:RequiredFieldValidator>
</td>
</tr>
<tr>
<td>重复密码：</td>
<td>
<asp:TextBox ID="txtRePassword"
  runat="server" TextMode="Password" MaxLength="20">
</asp:TextBox>
```

```
<br />
<asp:CompareValidator ID="cvPassword" runat="server"
  ControlToCompare="txtRePassword" ControlToValidate="txtPassword"
  ErrorMessage="两次密码需一致。">
</asp:CompareValidator>
</td>
</tr>
<tr>
<td>性别：</td>
<td>
<asp:RadioButtonList ID="rblIsMale" runat="server" RepeatColumns="2">
    <asp:ListItem Value="true" Selected>男</asp:ListItem>
    <asp:ListItem Value="false">女</asp:ListItem>
</asp:RadioButtonList>
</td>
</tr>
<tr>
<td>年龄：</td>
<td>
<asp:TextBox ID="txtAge" runat="server"></asp:TextBox></td>
</tr>
<tr>
<td>电话：</td>
<td>
<asp:TextBox ID="txtPhone" runat="server" MaxLength="20">
</asp:TextBox>
</td>
</tr>
<tr>
<td>邮箱：</td>
<td>
<asp:TextBox ID="txtEmail" runat="server" MaxLength="50">
</asp:TextBox>
</td>
</tr>
<tr>
<td>地址：</td>
<td>
<asp:TextBox ID="txtAddress" runat="server" MaxLength="255">
</asp:TextBox>
</td>
</tr>
<tr>
<td colspan="2">
<asp:Button ID="btnSubmit" runat="server" Text="提交"
  onclick="btnSubmit_Click" />
<asp:Button ID="btnLogin" runat="server" Text="返回登录"
  CausesValidation="false" PostBackUrl="Login.aspx" />
</td>
```

```
</tr>
</table>
```

没什么特别的东西，只是加了几个验证控件，RequiredFieldValidator 控件是验证非空的，CompareValidator 控件是比较两个控件值是否相同的。

这里“返回登录”按钮需要加一个 CausesValidation 属性，值设置为 flase，意思是单击时不触发验证。其他都见过，不再多说。

相关的代码如下：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    MemberInfo member = this.GetMemberByPage();
    //如果按会员名称查询能取得会员信息，说明该用户名已存在，不能再次注册
    if (MemberManager.GetMemberByLoginName(member.LoginName) != null)
    {
        Response.Write("<script>alert('用户名已存在，请换个用户名注册。');
                         location='Register.aspx';</script>");
        return;
    }
    MemberManager.CreateMember(member); //创建用户
    Response.Write("<script>alert('注册成功，请登录。');
                     location='Login.aspx';</script>");
}
private MemberInfo GetMemberByPage()
{
    MemberInfo member = new MemberInfo();
    member.LoginName = this.txtLoginName.Text;
    member.Password = this.txtPassword.Text;
    member.IsMale = bool.Parse(this.rblIsMale.Text);
    try { int.Parse(this.txtAge.Text); }
    catch { }
    member.Email = this.txtEmail.Text;
    member.Phone = this.txtPhone.Text;
    member.Address = this.txtAddress.Text;
    return member;
}
```

注册完以后，我们就可以用这个账号登录系统了。登录页面有一个文本框和一个密码框来接收用户输入。单击“登录”按钮后验证用户，下面是“登录”按钮的 onclick 事件处理程序：

```
protected void btnLogin_Click(object sender, EventArgs e)
{
    MemberInfo member =
      this.ValidateUser(this.txtLoginName.Text, this.txtPassword.Text);
    if (member != null) //如果验证成功，保存当前会员信息以记录登录状态
    {
        Session["CurrentUser"] = member;
        Response.Redirect("MemberCenter.aspx"); //跳转到用户中心页面
    }
```

```
    Response.Write("<script>alert('用户名或密码错误！！！');
                    location='Login.aspx';</script>");
}
private MemberInfo ValidateUser(string loginName, string password)
{
    /*
    * 验证用户名和密码，如果验证成功，返回该会员信息，否则返回 null
    */
    MemberInfo currentMember = MemberManager.GetMemberByLoginName(loginName);
    if (currentMember!=null && password==currentMember.Password)
    {
        return currentMember;
    }
    return null;
}
```

这里先从数据库取出会员信息来验证，如果验证成功，保存该对象到会话里并跳到会员中心页面，如果验证不成功，提示并返回重新登录。

登录以后，右上角就是一个以登录名命名的链接，点击后进入会员信息管理页面，可以修改除登录名外的所有信息，如图 14-13 所示。

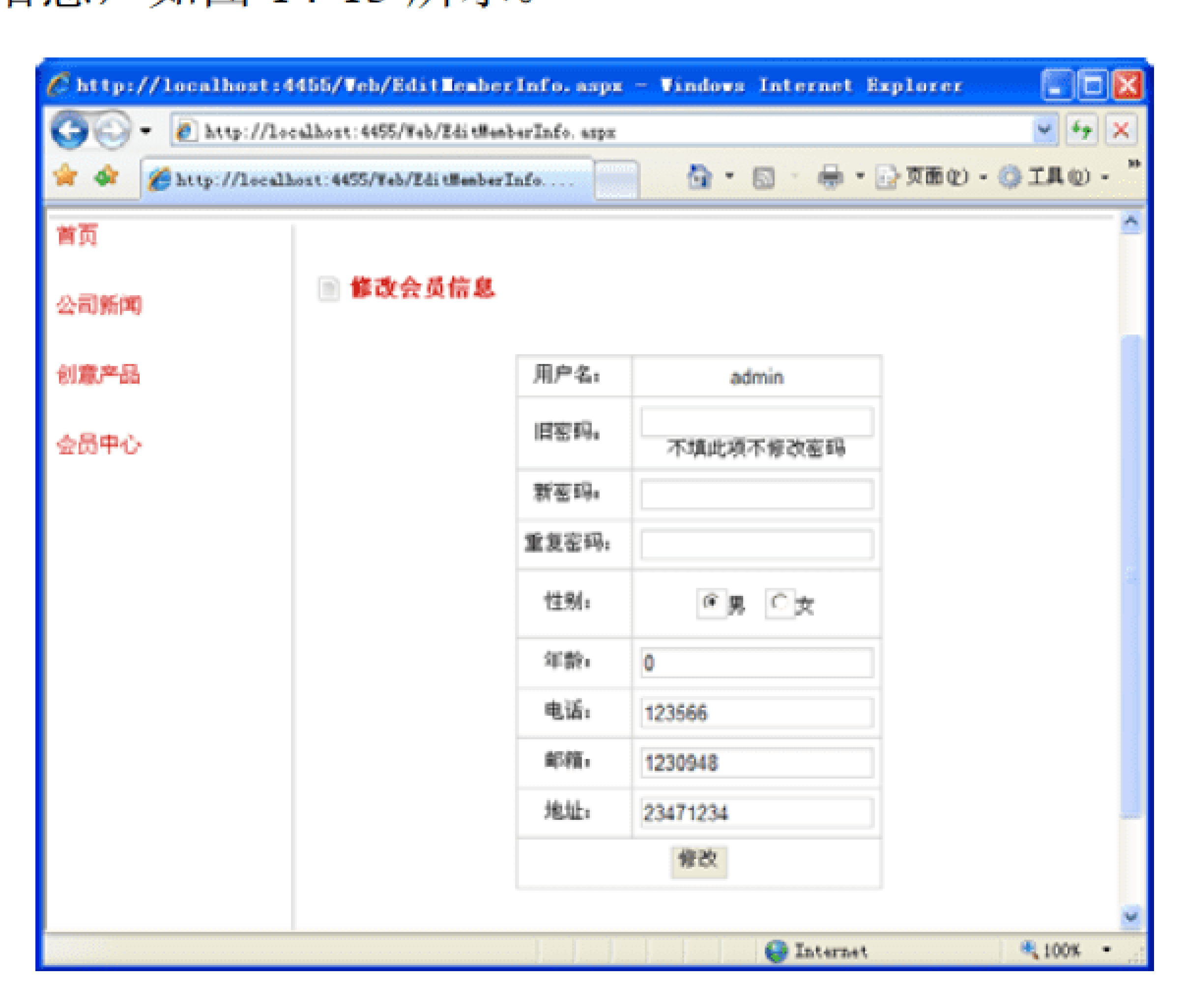


图 14-13　修改会员信息页面

如果不想修改密码，请保持“旧密码”文本框里为空。页面代码和注册页面相仿，只多加了一个“旧密码”文本框，不要那几个验证控件，这里不过多说明了。

下面来看一下后台代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.divResult.InnerHtml = "";
    this.ValidateUserPower();
    if (!IsPostBack)
    {
        this.FullPage();
```

```
    }
}
private void ValidateUserPower()
{
    if (Session["CurrentUser"] as MemberInfo == null)
        this.Response.Redirect("Login.aspx");
}
private void FullPage()
{
    /*
     * 取出当前用户信息，填充至页面控件
     */
    MemberInfo member = Session["CurrentUser"] as MemberInfo;
    this.lblLoginName.Text = member.LoginName;
    this.txtPhone.Text = member.Phone;
    this.txtEmail.Text = member.Email;
    this.txtAge.Text = member.Age.ToString();
    this.txtAddress.Text = member.Address;
    //页面控件里的值是true或false，所以这里要转换成小写
    this.rblIsMale.SelectedValue = member.IsMale.ToString().Trim().ToLower();
}
```

照例还得先绑定对象信息到页面控件，但是注意一点——这里验证了一下用户权限，也就是说看当前会话有没有会员登录信息，如果没有，肯定不让继续访问，跳转到登录页面：

```
protected void btnSubmit_Click(object sender, EventArgs e)
{
    MemberInfo member = this.GetMemberForPage();
    switch (this.ValidateOldPassword())
    {
        case 0:
            //这里和业务逻辑层约定，如果密码为null，就不修改密码
            member.Password = null;
            break;
        case 1:
            this.divResult.InnerHtml = "<script>alert('旧密码输入错误。');</script>";
            return;
        case 2:
            if (this.txtPassword.Text == string.Empty)  //判断输入的新密码是否为空
            {
                this.divResult.InnerHtml = "<script>alert('新密码不能为空。');</script>";
                return;
            }
            else if (this.txtRePassword.Text != this.txtPassword.Text)
              //判断两次新密码是否一致
            {
                this.divResult.InnerHtml = "<script>alert('两次密码必须一致。');</script>";
                return;
            }
```

```
            else
            {
                member.Password = this.txtPassword.Text;
            }
            break;
    }
    MemberManager.ModifyMemberInfo(member);
    Session["CurrentUser"] =
      MemberManager.GetMemberByLoginName(member.LoginName);
    this.divResult.InnerHtml = "<script>alert('保存成功。');</script>";
}
private MemberInfo GetMemberForPage()
{
     /*
      * 将页面值封装成对象
    */
    MemberInfo member = new MemberInfo();
    member.LoginName = this.lblLoginName.Text;
    member.Phone = this.txtPhone.Text;
    member.IsMale = bool.Parse(this.rblIsMale.SelectedValue);
    try { member.Age = int.Parse(this.txtAge.Text); }
    catch { }
    member.Address = this.txtAddress.Text;
    member.Email = this.txtEmail.Text;
    return member;
}
private int ValidateOldPassword()
{
    //0 旧密码为空，不修改密码；1 密码不正确，2 验证通过
    string oldPass = this.txtOldPassword.Text;
    string loginName = this.lblLoginName.Text;
    if (oldPass.Length <= 0)     //如果旧密码为空
    {
        return 0;
    }
    else
    {
        //从数据库取出当前用户信息
        MemberInfo member = MemberManager.GetMemberByLoginName(loginName);
        if (oldPass != member.Password)  //如果密码输入不正确
        {
            return 1;
        }
        else
        {
            return 2;
        }
    }
}
```

代码有点长，不过没关系，我们一点点地看。

总体思路是这样的：如果旧密码为空，说明不修改密码；如果旧密码不为空，但输入不正确，就提示，并返回；如果旧密码输入正确，就可以测试新密码了，这种情况下如果新密码没有输入，提示并返回，如果两次输入的不一致，也提示并返回，如果两次输入都没有问题，就可以更新数据库，更新完数据库顺便也更新一下当前会话，以便保持最新信息，最后提示保存成功。顺着这个思路慢慢捋，整个程序结构就清晰了。

当然，业务逻辑层也不能啥都不干：

```
public static void ModifyMemberInfo(MemberInfo member)
{
    //取出数据库里未修改的会员信息
    MemberInfo oldMember = MS.GetMemberByLoginName(member.LoginName);
    member.ID = oldMember.ID;
    //如果密码为 null，表示不修改密码把旧密码赋给新对象
    if (member.Password == null) member.Password = oldMember.Password;
    MS.Update(member);  //用新对象更新数据库
}
```

先从数据库读出旧的会员信息，如果密码不需要修改，就把旧密码赋给新对象，再用新对象更新数据库就行了。

14.9.5 订单管理

刚才说了，在查看产品的时候，可以点“订购该产品”来下订单，不过前提是必需登录系统，否则会跳到登录页面的。这里假设登录过了，就来到创建订单的页面了，如图 14-14 所示。



图 14-14 创建订单页面

这里只要简单地填上用户要订购的数量，还有其他一些说明，就可以单击“订购”按钮提交订单了。

我们来看是怎么实现的。

页面上用到两个 Label 和两个 Textbox，比较简单，就不贴代码了。

首先还是得初始化页面，给用户尽可能人性化的提示信息。这里再次提示用户订购的东西的名字和价格。下面是数据绑定代码：

```
private void BindProduct()
{
    int id = this.GetProductId();   //获取产品 ID
    ProductInfo product =
      ProductManager.GetProductById(id);    //从数据库取出产品信息
    if (product == null)
       Response.Redirect("ProductList.aspx"); //如果不存在该产品信息，返回列表
    this.lblPrice.Text = product.Price.ToString();
    this.lblProductName.Text = product.Name;
}
private int GetProductId()
{
    int id = 0;
    try { id = int.Parse(Request["id"]); }
    catch { }
    return  id;
}
```

同样，为了防止邪恶事情的发生，必需验证数据安全性。其实还得验证一下用户权限，看会员是否登录，前面介绍过，这里就略过。

前面只是把订单页面呈现了出来，下面开始实现提交订单功能：

```
private int GetCurrentUserId()
{
    MemberInfo currentMember = Session["CurrentUser"] as MemberInfo;
    return currentMember.ID;
}
private OrderInfo GetOrderByPage()
{
    OrderInfo newOrder = new OrderInfo();
    newOrder.ProductID = this.GetProductId();   //产品 ID 为提交过来的 ID
    newOrder.MemberID = this.GetCurrentUserId();   //会员 ID 为当前登录账号 ID
    try { newOrder.Number = int.Parse(this.txtNumber.Text); }
    catch { }
    newOrder.Details = this.txtDetails.Text;
    newOrder.PublishTime = DateTime.Now;
    return newOrder;
}
protected void btnSubmit_Click(object sender, EventArgs e)
{
    OrderManager.CreateOrder(this.GetOrderByPage());
    Response.Write("<script>alert('提交成功，点击确定回到产品列表。');
                    location='ProductList.aspx'; </script>");
}
```

这里获取提交过来的产品 ID、当前登录用户的 ID 和当前时间加上用户输入的信息，创建订单对象，由业务逻辑层传递给数据访问层，保存到数据库。这里用户输入的“订购数量”可能不是合法的阿拉伯数字，所以要捕获一下异常，至于捕获后的异常处理，这里就省略了。

下面该说查看订单了。查看订单没太多复杂的东西，这里只需要将所有订单列出来就行了，如图 14-15 所示。



图 14-15　订单列表页面

还是老规矩，使用 Repeater：

```
<table id="order_list" cellspacing="1">
<asp:Repeater ID="rptOrders" runat="server">
<ItemTemplate>
<tr>
    <td>产品名称：<%# GetProductNameById(Eval("ProductID"))%></td>
    <td>数量：<%# Eval("Number") %></td>
    <td>时间：<%# Eval("PublishTime") %></td>
</tr>
<tr>
    <td colspan="3">
    详细说明：<%# Eval("Details") %><br /><br />
    <div style='text-align:right;
      <%# bool.Parse(Eval("Status")+"")?"":"display:none" %>'>已处理</div>❶
    </td>
</tr>
</ItemTemplate>
</asp:Repeater>
</table>
```

❶这句可能看着有点不太习惯，如果订单的状态为 false，说明订单未处理，页面就不再显示该 div。

下面是页面要用到的辅助方法：

```
protected string GetProductNameById(object oid)
{
    ProductInfo product =
      ProductManager.GetProductById(int.Parse(oid.ToString()));
    return product.Name;
}
```

同样，需要绑定 Repeater 的数据源。这里先取得当前用户，根据当前用户 ID 取得一个数据列表作为 Repeater 的数据源对象：

```
private void BindOrder()
{
    MemberInfo member = Session["CurrentUser"] as MemberInfo;
    if (member != null)
    {
        //根据当前登录用户 ID 取出当前用户的订单信息
        this.rptOrders.DataSource = OrderManager.GetOrderListByMemberID(member.ID);
        this.rptOrders.DataBind();
    }
}
```

14.10　总　　结

在做以前，可能我们会觉得可能会有多少困难、多少未知因素，不过经过上面的一通叙述，是不是已经有思路了呢？

其实信息管理系统说白了就是获得用户输入，存到数据库里，再读出来展示一下，不要觉得有多大困难。

细心的或者有经验的读者或许已经发现，上面的例子是一个残缺不全的半成品。不过这只是入门学习的实例，不可能尽善尽美。

当然，发现问题就立即解决它，这是一个很好的学习习惯。比如读者可以在这个实例的列表部分加上分页功能，让它变得更加实用。



第 15 章　鲜花预订系统

内容摘要：

鲜花预订系统可以让顾客足不出户，直接在网上进行鲜花预订，并接收用户在线订单。本章实例采用 ASP.NET + C# + ADO.NET 技术实现简单的鲜花预订系统功能。这个系统基于.NET Framework 平台，使用 C#语言操作 ASP.NET WebForm 和 ADO.NET。开发工具选择 VS2008，数据库选择 SQL Server 2008，实现了鲜花订购、订单管理、用户管理等基本功能。

学习目标：

- 了解鲜花预订系统的功能概述和操作流程
- 理解鲜花预订系统的架构设计和功能模块
- 掌握系统数据库的设计和实现过程
- 掌握公共模块的实现方法
- 掌握查看、添加和删除鲜花功能的实现
- 掌握系统中购物车和订单的实现

15.1 系 统 概 述

本系统为网上鲜花预订系统。目前送花已经成为了一种时尚，当人们无法会见朋友时，会委托花店为指定的某个地址的某人送一束鲜花。本系统实现一套鲜花预订管理系统，用户在该系统上可以预订鲜花和查询鲜花信息等。鲜花预订系统给用户提供了方便的网络平台。预订的用户通过订单将自己需要的鲜花种类、数量、送货地址和付费方式表述清楚，花店根据收到的订单信息向用户提供鲜花配送服务。

15.1.1 系统功能

本系统在前台为普通用户提供了预订和查询鲜花资讯的功能，在后台为花店管理员提供了对系统平台的管理功能。

在前台为用户提供的操作包括以下内容：个人信息管理、用户订单管理、购物车和查找功能。此系统为管理员提供的功能如下：对普通用户的管理、鲜花信息的管理、用户的订单管理。

15.1.2 系统预览

如图 15-1 所示为游客进入本网站后的主界面。从主界面可以浏览到本站的所有鲜花以及查看鲜花信息，但游客不能对鲜花进行订购，只能查看鲜花。



图 15-1 游客界面

如图 15-2 所示为管理员登录网上花店后的界面，管理员可操作的模块有用户管理、鲜花管理、订单管理、查找鲜花及退出系统。

如图 15-3 所示为会员登录网上花店后的界面，会员的功能模块有查看订单、查找鲜花、

修改个人密码、退出系统的功能。



图 15-2　管理员界面



图 15-3　会员界面

15.1.3　系统操作流程

本系统首先展示给用户的是 Home.aspx 界面，即网站的首页，此界面均为用户自定义控件所组成，游客只能浏览到本站的鲜花，不能进行购买，需要注册为会员才可进行购买，在登录界面通过验证用户的登录身份确认用户为会员还是管理员，从而进入相应的界面。如果判断是管理员，展现给用户的就是用户管理、鲜花管理、订单和查找功能模块；如果是会员，则展示

其购物车、个人订单、修改密码、查找鲜花功能模块。

鲜花预订系统的操作流程如图 15-4 所示。

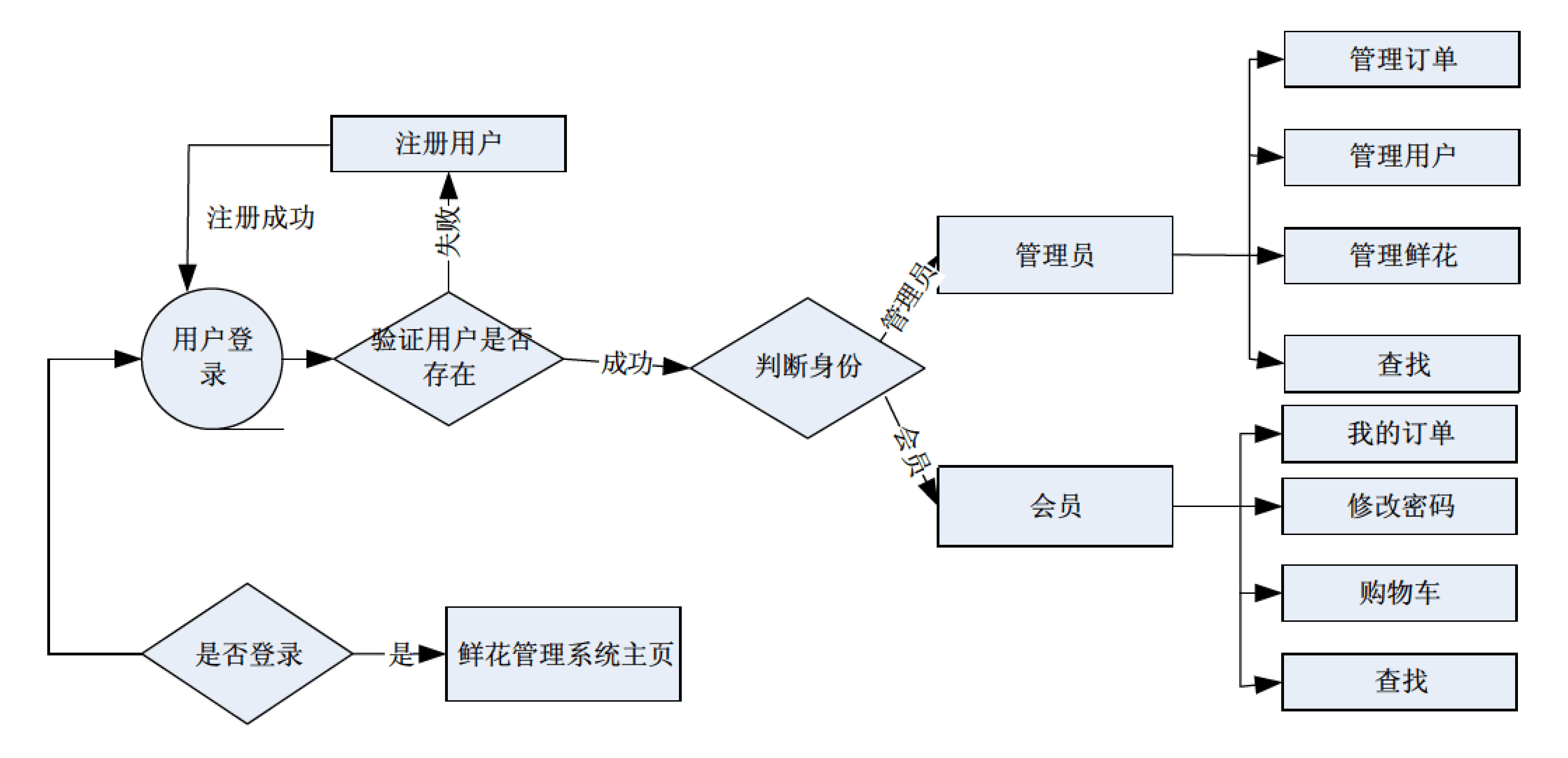


图 15-4　鲜花预订系统的操作流程

15.2　系统设计架构

本系统采用两层架构，页面通过 Web 层对数据库直接访问，不使用中间业务逻辑层以及与数据库连接的接口。数据库为系统的底层，其中数据访问层封装了数据库的选择、更新、删除、添加等基本操作，同时还为 Web 展示层提供数据库访问的接口和函数。

15.2.1　系统架构

本系统采用 ASP.NET 应用程序最基本的两层架构，其结构如图 15-5 所示。各层功能的介绍如表 15-1 所示。

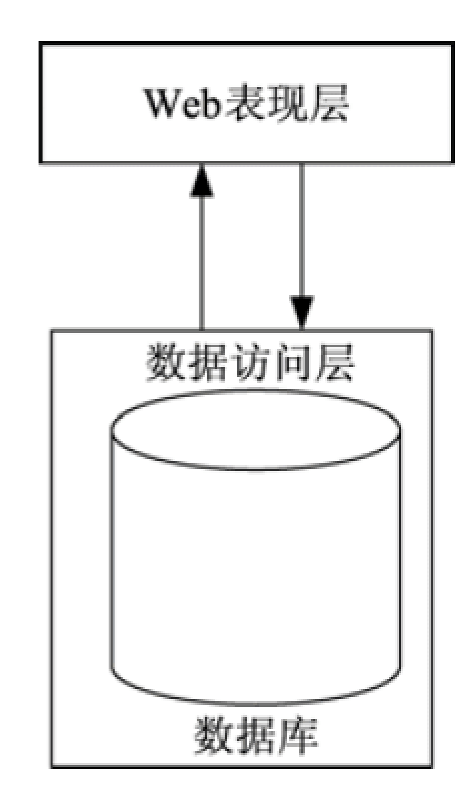


图 15-5　两层架构模式

表 15-1　两层架构模式各层的功能

层　名	功能描述
Web 表现层	位于系统最顶层，向用户展示各种界面，实现用户各种操作信息的添加、删除和修改。与数据访问层直接关联，其数据访问模块封装了对数据库的所有操作
数据库	用来存储本系统的所有数据信息

15.2.2　系统功能模块

根据上面的分析，可以画出系统的功能模块图，分为管理员和普通用户两个大的模块，具体模块划分如图 15-6 所示。

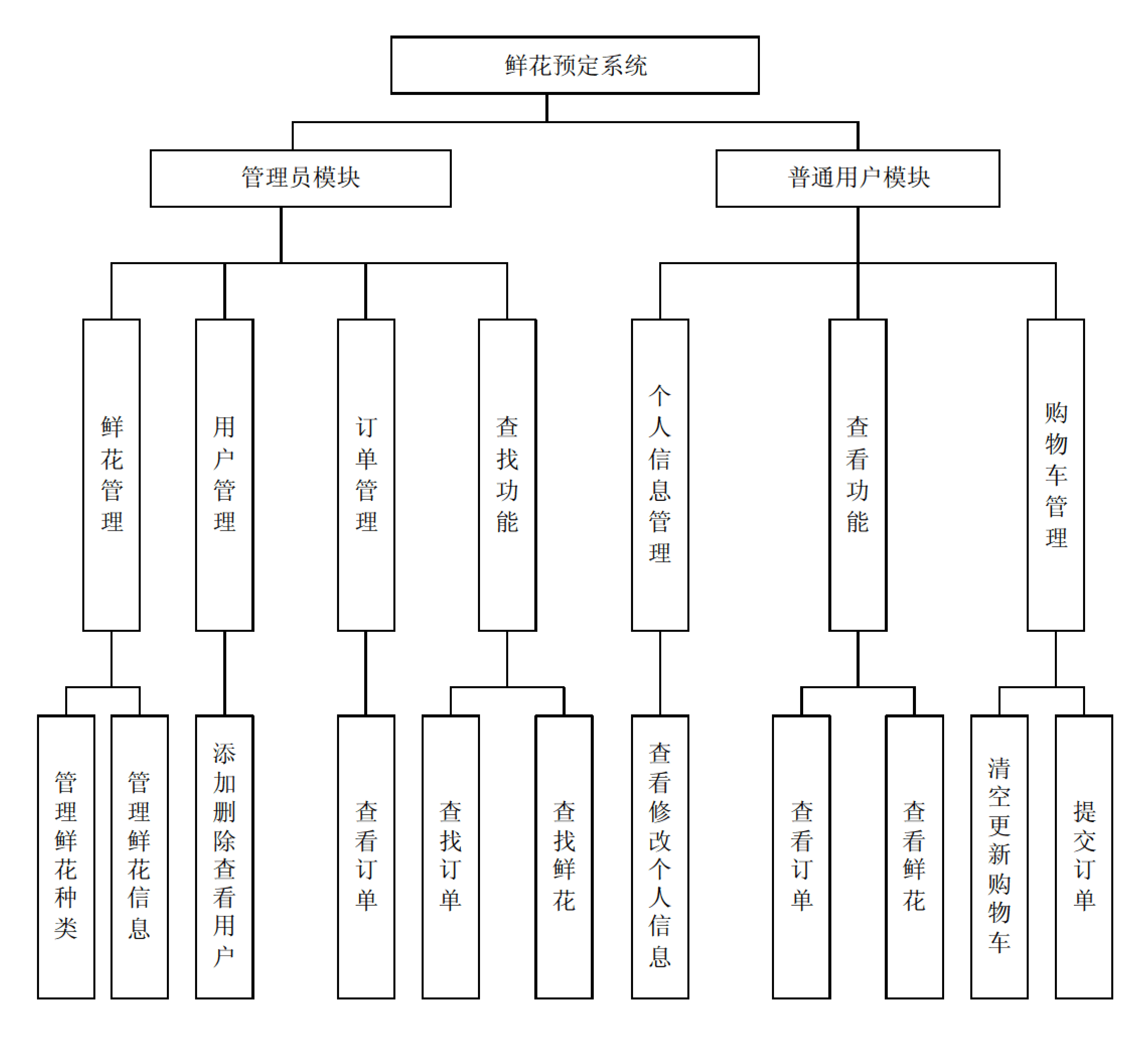


图 15-6　系统功能模块划分

15.3 数据库的设计和实现

这一节主要讲述数据库的结构和实现思想。下面将针对数据库的分析和设计思路进行详细描述。

15.3.1 数据库需求分析

(1) 用户的具体需求为各个信息的提供、保存、更新与查询，对于此系统有如下需求：

- 用户分为管理员用户和普通用户，通过一个 ID 进行区分。
- 普通用户可以有多个订单。
- 一个订单可以有多种鲜花，一个鲜花也可以被多个订单订购。
- 管理员可以对用户进行增删查改，也可以对鲜花进行管理。
- 不是已注册用户不能进行鲜花购买。

(2) 经过上述总结，设计如下数据项。

- 用户信息表：用户 ID、用户名、密码、权限 ID、地址、E-mail、联系电话。
- 鲜花信息表：鲜花 ID、类别 ID、鲜花名称、鲜花数量、鲜花图片、鲜花价格、折后价格、销售数目、点击数目。
- 订单信息表：订单 ID、购买者姓名、所购鲜花名称、所购鲜花数量、单价、总价。
- 网站新闻表：新闻 ID、新闻标题、新闻内容。
- 鲜花库存表：库存 ID、类别名称、备注。
- 网站顶部菜单表：菜单 ID、菜单名称、链接 URL。

15.3.2 数据库概念结构设计

由上面的数据项可以设计出能够满足用户需求的各种实体，以及它们之间的关系，为后面的逻辑结构设计打下基础。这些实体包含各种具体信息，通过相互之间的作用形成数据的流动，这样就可以对本系统设计以下实体：用户信息实体、鲜花实体、订单信息实体、鲜花库存实体。

(1) 用户信息实体的 E-R 图如图 15-7 所示。

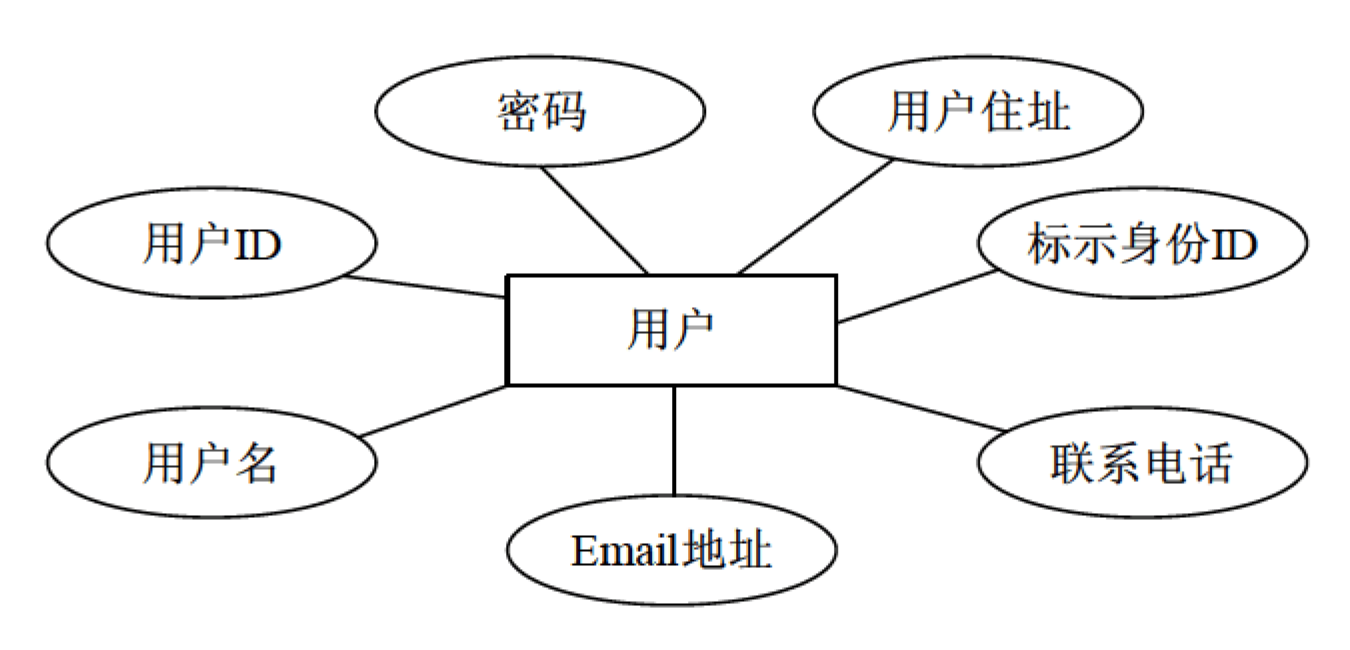


图 15-7 用户实体的E-R图

(2) 鲜花信息实体的 E-R 图如图 15-8 所示。

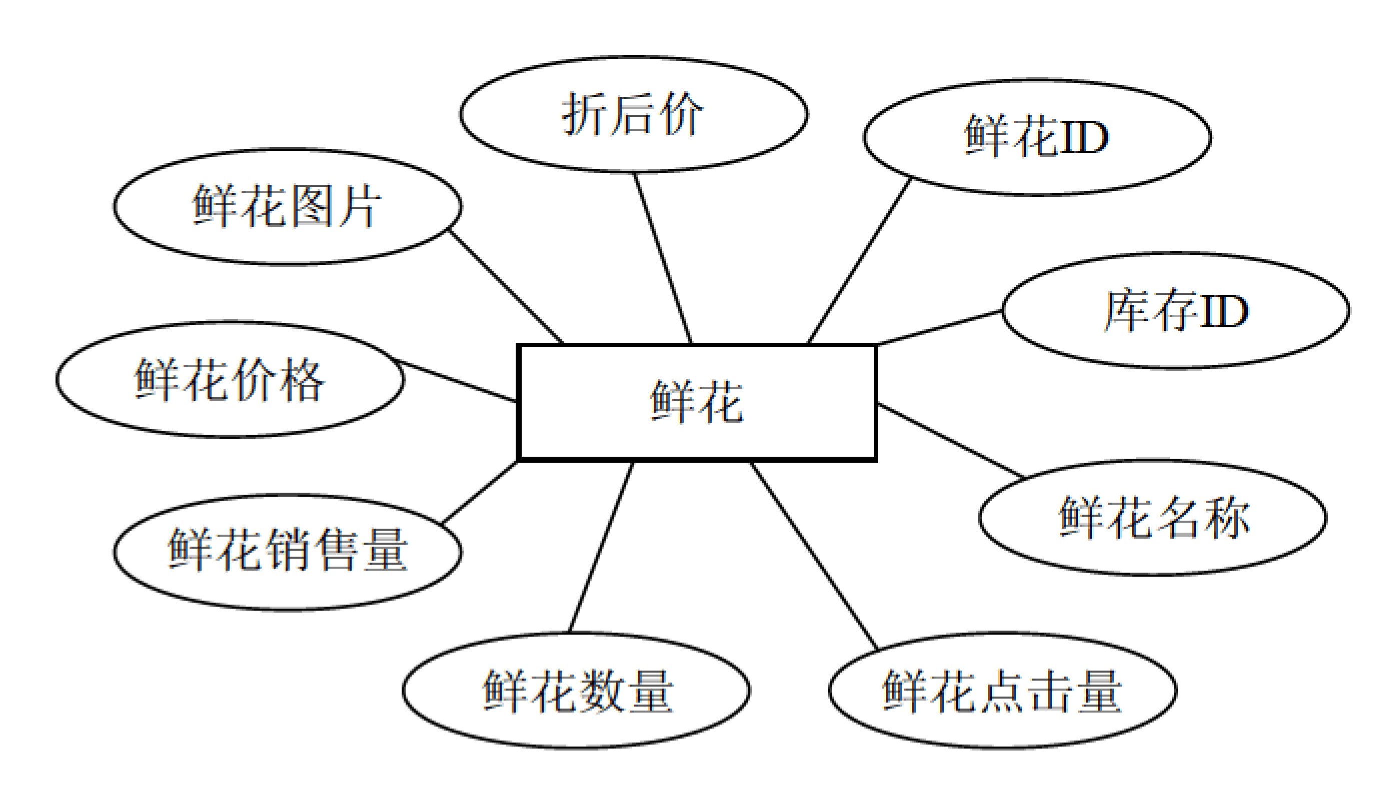


图 15-8　鲜花实体的E-R图

(3) 订单实体的 E-R 图如图 15-9 所示。

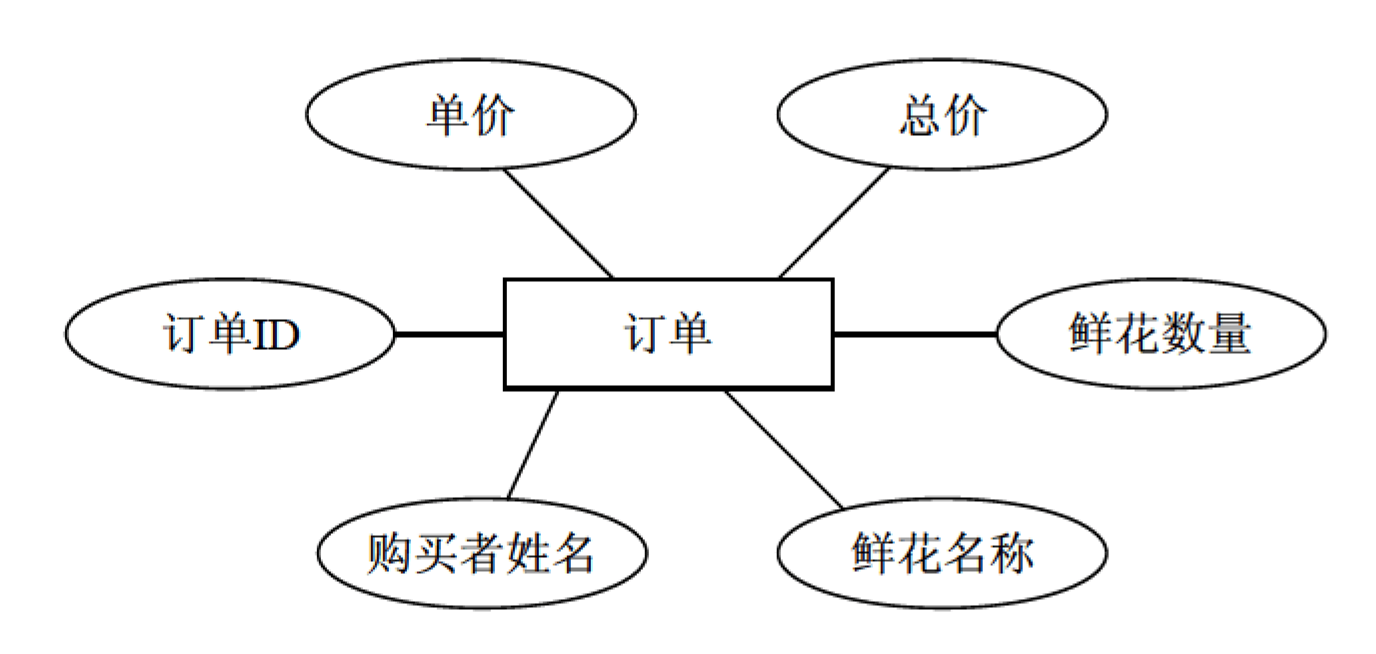


图 15-9　订单实体的E-R图

(4) 鲜花库存信息实体的 E-R 图如图 15-10 所示。

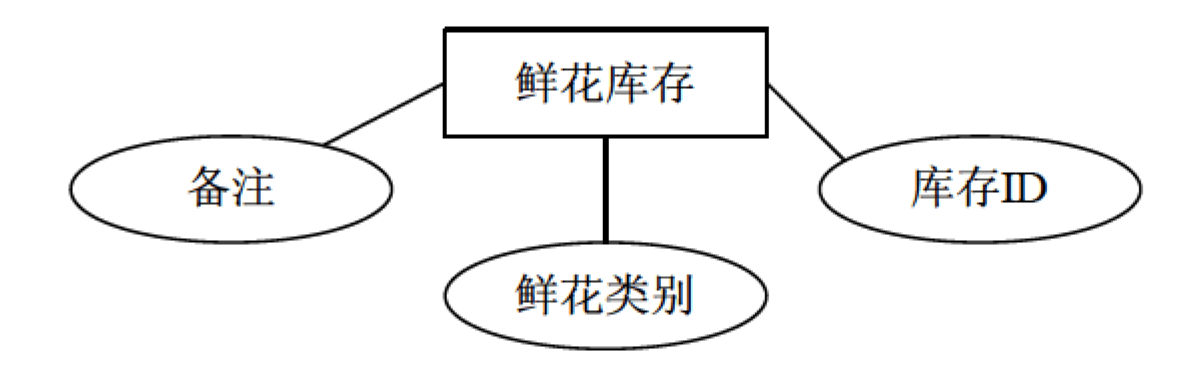


图 15-10　鲜花库存信息实体的E-R图

15.3.3　数据表设计

本系统采用 MS SQL Server 2008 数据库系统，首先我们建立一个名为 BuyDataBase 的数据库。这里数据库登录账号使用的是系统账号 sa，密码是 123456，下面是该数据库的结构信息。根据上面的需求分析，鲜花预订系统数据库中的各个表的设计结果如下面各个表格所示，每个表格表示数据库中的表。

1. 用户信息表(users)

该表包含用户编号、用户名、用户密码、身份 ID、用户住址、用户 E-mail、联系电话这 7 个字段，如表 15-2 所示。

表 15-2 用户表

字 段	数据类型	长 度	允 许 空	备 注
userid	INT	4	否	用户编号
username	NVARCHAR	50	否	主键，用户名
password	NVARCHAR	50	否	用户登录密码
limitid	INT	4	否	标示用户身份 ID
address	NVARCHAR	50	否	用户住址
email	NVARCHAR	50	否	用户 E-mail 地址
phonenum	NVARCHAR	50	是	用户联系电话

2. 鲜花库存表(category)

该表包含库存编号、鲜花类别名、备注这 3 个字段，如表 15-3 所示。

表 15-3 鲜花库存表

字 段	数据类型	长 度	允 许 空	备 注
id	INT	4	否	主键，库存编号
catename	NVARCHAR	50	否	鲜花类别名称
demo	NVARCHAR	50	是	备注

3. 鲜花信息表(flower)

该表包含鲜花编号、库存编号、鲜花名称、鲜花数量、鲜花图片、鲜花单价、鲜花折后价、已售出数目、已被查看数这 9 个字段，如表 15-4 所示。

表 15-4 鲜花信息表

字 段	数据类型	长 度	允 许 空	备 注
flowerid	INT	4	否	鲜花编号
categoryid	INT	4	否	所属库存编号
flowername	NVARCHAR	50	否	主键，鲜花名称
flowernum	INT	4	否	鲜花数量
flowerimage	VARCHAR	50	否	鲜花图片
price	MONEY	8	否	鲜花单价
disprice	MONEY	8	否	鲜花折后价
sale	INT	4	否	已售出
spare	INT	4	否	已被查看数

4. 菜单信息表(menu)

该表包含菜单编号、菜单名称、菜单 URL 地址这 3 个字段，如表 15-5 所示。

表 15-5　菜单信息表

字　段	数据类型	长　度	允 许 空	备　注
id	INT	4	否	主键，菜单编号
name	NVARCHAR	50	否	菜单名称
URL	NVARCHAR	50	是	菜单链接的 URL 地址

5. 订单信息表(shoppingcart)

该表包含订单编号、鲜花名称、鲜花数量、鲜花单价、鲜花总价、用户名这 6 个字段，如表 15-6 所示。

表 15-6　订单信息表

字　段	数据类型	长　度	允 许 空	备　注
id	INT	4	否	主键，订单编号
flowername	NVARCHAR	50	否	鲜花名称
flwnum	INT	4	否	鲜花数量
price	MONEY	8	否	鲜花单价
total	MONEY	8	否	鲜花总价
username	NVARCHAR	50	否	用户名

6. 新闻信息表(news)

该表包含新闻编号、新闻标题、新闻内容这 3 个字段，如表 15-7 所示。

表 15-7　新闻信息表

字　段	数据类型	长　度	允 许 空	备　注
id	INT	4	否	主键，新闻编号
name	NVARCHAR	50	否	新闻标题
text	NVARCHAR	50	否	新闻内容

15.3.4　数据表之间的关系

数据库各个表之间的关系如图 15-11 所示。

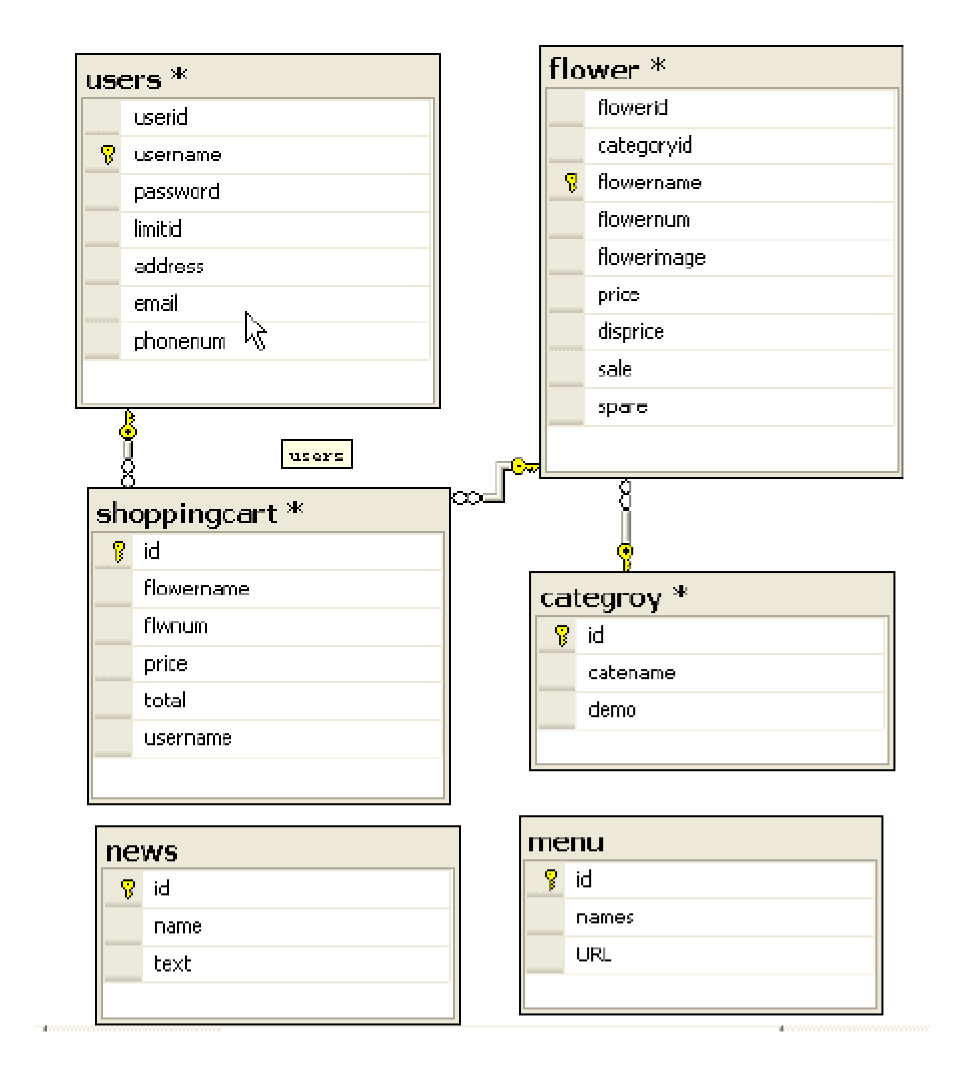


图 15-11　数据表之间的关系

15.4　公用模块的编写

在网站构建中，不免会有许多的页面和功能的实现需要用到公共的部分。因此，我们可以将这些都运用到的部分作为一个模块，在需要时采取“拿来主义”，而不必每当用到时重新构建这些重复使用的模块。

15.4.1　数据库连接的编写

为了方便应用程序的移植，此系统在应用程序的配置文件(Web.Config)中设置数据库连接信息。添加如下语句：

```
<configuration>
   <connectionStrings>
      <add name="constr" connectionString="Data Source=.;
        Initial Catalog=BuyDataBase;User ID=sa;password=123456"/>
   </connectionStrings>
   ...
</configuration>
```



上述代码中的 User ID 和 password 应该与本机上的 SQL Server 的登录名和密码相互对应。

15.4.2　界面主体框架

此网站应用了母版页技术，各个页面为不同的内容页，内容页由不同的用户控件组成，这

样整个系统的构建就非常简单了，只需根据不同的用户请求，给用户展示不同的界面就可以了，接下来就来看母版页的实现吧。

母版页为公共页面的使用部分，用户可以在母版页上设置常用菜单项，也可以为整个系统页面布局。鲜花预订系统的母版页使用 DIV+CSS 布局，CSS 文件 StyleSheet.css 的代码如下：

```
body
{
    background-color: #f5e1fb;
}
.td_css
{
    background-image: url("image/BannerBGBlue.gif");
    height: 24px;
}
:link
{
    color: Green;
}
:visited
{
    color: Gray;
}
:active
{
    color: Red;
}
er
{
    color: Blue;
}

table
{
    background-color: #f5e1fb;
    height: 63px;
}
.style1
{
    height: 22px;
    width: 654px;
}
.style3
{
    width: 402px;
}
.style4
{
    height: 30px;
```

```
    width: 654px;
}
.style5
{
    width: 654px;
}
```

结合各个用户自定义控件和上述定义的样式表文件，母版页的设计视图如图 15-12 所示。

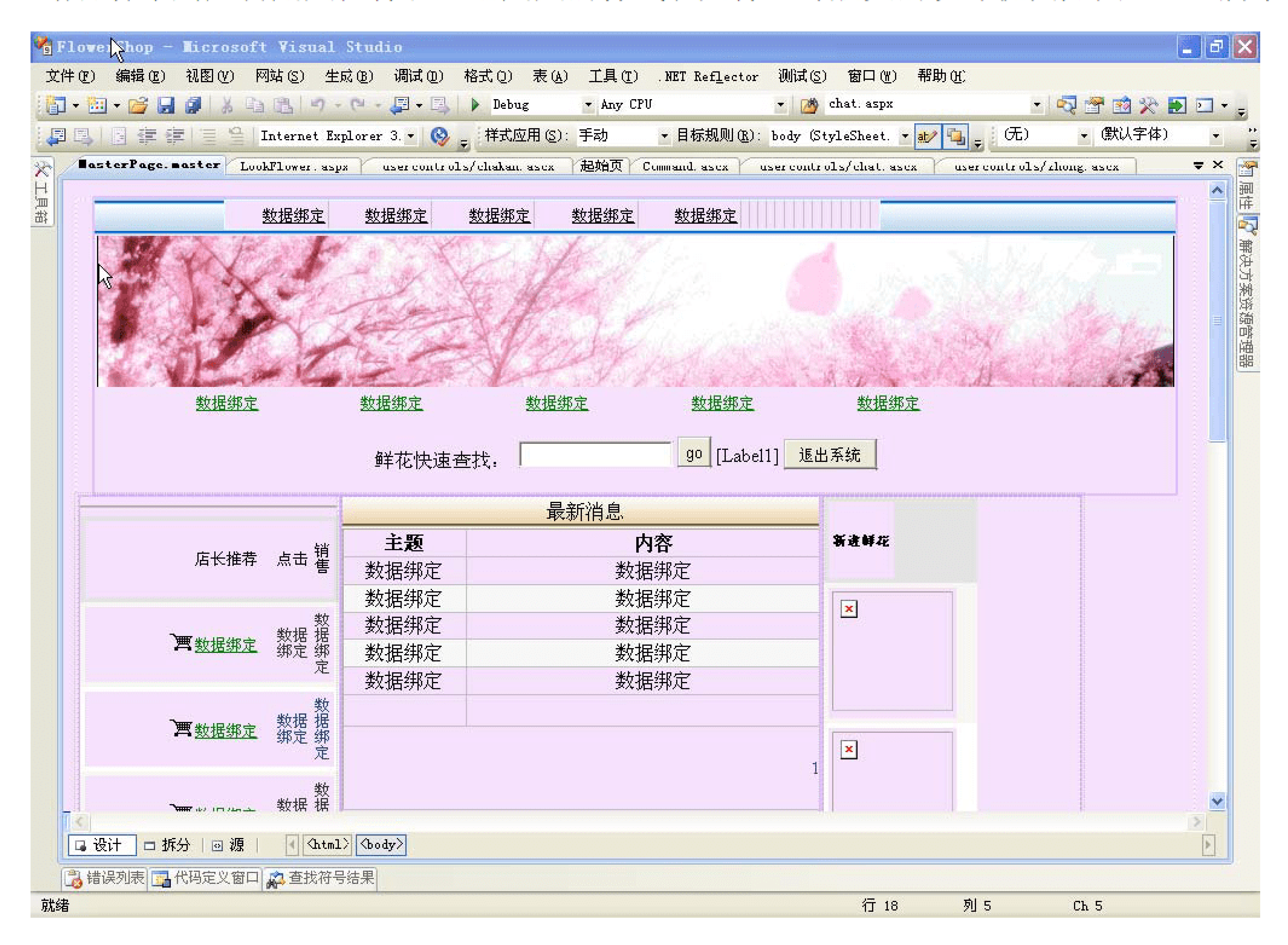


图 15-12　母版页的设计视图

15.4.3　页面通用模块

为了提高主要代码的重复使用性，本系统运用了母版页和用户自定义控件，每个页面均为内容页，根据用户不同的需求从而显示不同的页面。其中每个页面都重用到的用户自定义控件为 header.ascx(头部)和 foorer.ascx(底部)，其中头部控件如图 15-13 所示。

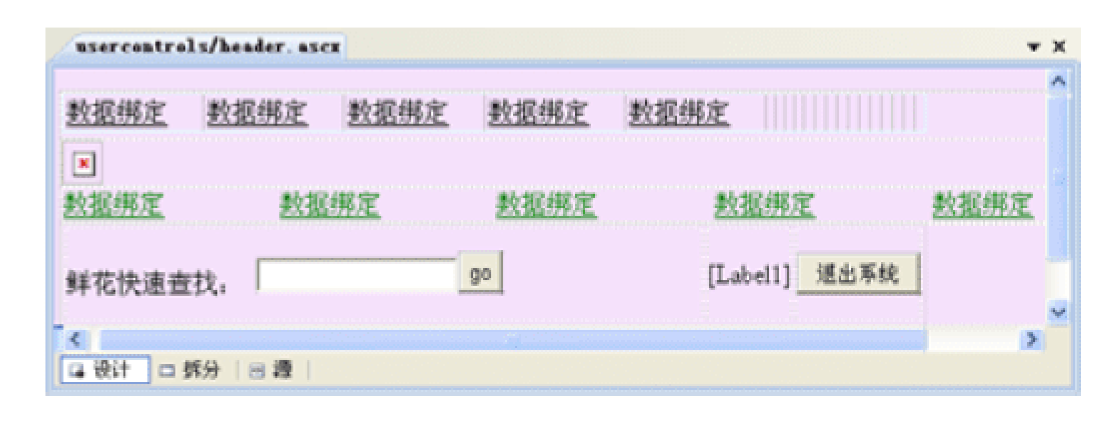


图 15-13　头部控件设计视图

在图 15-13 中，头部控件运用的控件有：两个 DataList 控件、两个 Label 控件、两个 Button 控件和一个 TextBox 控件。其页面代码部分如下：

```
<table border="0" cellpadding="1" cellspacing="0" align="center"
  style="width: 89%; height: 89px">
    <tr>
        <td valign="top" style="background-image:
            url('image/BannerBGBlue.gif'); background-position: 50% bottom;"
            class="style4">
            <asp:DataList ID="DataList1" runat="server" Height="8px"
                BorderColor="LightCyan"
                RepeatColumns="20" RepeatDirection="Horizontal" CellPadding="2"
                GridLines="Both"
                Width="547px">
                <ItemTemplate>
                    <asp:HyperLink ID="HyperLink1" runat="server"
                    NavigateUrl='<%# Eval("URL")%>' Text='<%# Eval("names") %>'
                    Font-Underline="True" ForeColor="Black"
                    Font-Overline="False" BorderColor="#666699"
                    Font-Size="Small" Font-Bold="False"></asp:HyperLink>
                </ItemTemplate>
            </asp:DataList>
        </td>
    </tr>
    <tr>
        <td class="style5">
            <img id="toppic" src="./image/header.gif" align="middle" />
        </td>
    </tr>
    <tr>
        <td class="style5">
            <asp:DataList ID="DataList2" runat="server"
                RepeatDirection="Horizontal" Height="16px"
                Width="688px">
                <ItemTemplate>
                    <asp:HyperLink ID="HyperLink2"
                     NavigateUrl='<%# "~/ShowFlower.aspx?id=" + Eval("id")%>'
                     runat="server" Text='<%# Eval("catename") %>'
                     Font-Size="Small" Font-Bold="False"></asp:HyperLink>
                </ItemTemplate>
            </asp:DataList>
        </td>
    </tr>
    <tr>
        <td class="style1">
            <table>
                <tr>
                    <td class="style3">
                        鲜花快速查找：
```


Web 程序员成功之路

```
                    <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
                    <asp:Button ID="Button1" runat="server" Text="go"
                     OnClick="Button1_Click" />
                </td>
                <td align="center">
                    <asp:Label ID="Label1" runat="server"></asp:Label>
                </td>
                <td>
                    <asp:Button ID="btn_Out" runat="server" Text="退出系统"
                     OnClick="btn_Out_Click" />
                </td>
            </tr>
        </table>
    </td>
  </tr>
</table>
```



在上述的页面代码中，需要添加 StyleSheet.css 样式文件。

在 Page_Load 事件中，添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Session["username"].ToString() != "")
        {
            this.Label1.Text =
              Session["username"].ToString(); //显示当前的用户名
        }
        else
        {
            this.Label1.Text = "欢迎您光临本花店，愿你选购开心~-~";
        }
        Getdlist1();
        Getdlist2();
    }
}
```

上述代码中 Label 控件所显示的为当前用户登录的用户名，它是使用 Session 保存的。使用 Session 变量的好处是它能随着用户的退出而自动销毁。

其中 GetList1()方法的代码如下：

```
public void Getdlist1()
{
    string s = "select * from menu"; //查询数据库中的菜单表
    DataSet ds = sf.GetDataSet(s); //作为 DataSet 返回
    DataList1.DataSource = ds.Tables[0].DefaultView; //设置数据源
    DataList1.DataBind(); //进行绑定
```

```
}
```

对页面的 DataList 数据控件所绑定的均是从数据库查询出的 DataSet 数据集，取得数据后直接绑定到 DataList 控件即可。

与上述代码类似，其中 GetList2()方法的代码如下：

```
public void Getdlist2()
{
    string s = "select top 10 * from categroy"; //查询数据库中的库存表
    DataSet ds = sf.GetDataSet(s); //查询结果作为 DataSet 返回
    DataList2.DataSource = ds.Tables[0].DefaultView; //设置数据源
    DataList2.DataBind(); //进行绑定
}
```

在文本框中输入所要查询的鲜花名称，再单击 go 按钮，即可进行查询。双击 go 按钮，进入其 Click 事件，添加如下代码：

```
protected void Button1_Click(object sender, EventArgs e)
{
    string str = TextBox1.Text.Trim(); //获取 TextBox 中的值
    if (str != null)
    {
        Server.Transfer("LookFlower.aspx?id=" + str, true); //跳转页面
    }
    else
    {
        Response.Write(
          "<script type='text/javascript'>alert('文本框不能为空！')</script>");
    }
}
```

上述代码首先定义一个字符串来接受用户输入到文本框的内容，然后把它作为参数 id 的值传给另一个页面(LookFlower.aspx)。注意要先判断文本框内的值不能为空。

双击“退出系统”按钮，进入其 Click 事件，添加如下代码：

```
protected void btn_Out_Click(object sender, EventArgs e)
{
    Session["username"] = ""; //将 Session 清空
    Response.Redirect("Home.aspx"); //跳转到主页面
}
```

退出系统时，需要将 Session 的值清空。至此，头部自定义控件(header.ascx)构建完毕。

底部控件如图 15-14 所示。

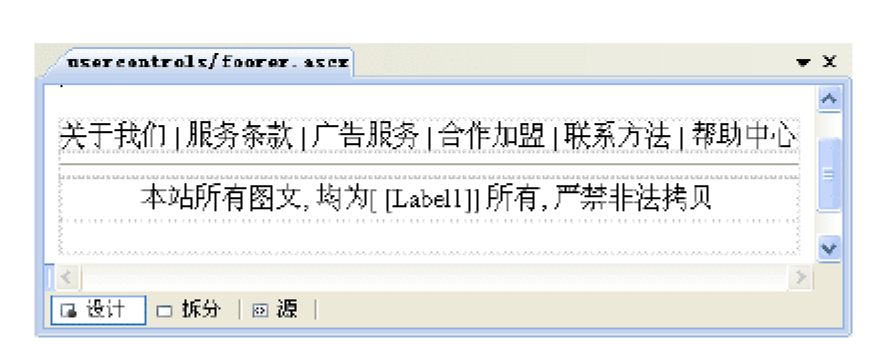


图 15-14　底部控件设计视图

在图 15-14 中，底部控件 footer.ascx 中仅有一个 Table，具体代码这里就不再介绍，可参考源文件。

15.4.4　登录系统和退出系统

在进行鲜花购买时，系统会判断当前为游客还是会员，如是游客，则提供免费注册功能，如是会员，则判断此会员是否已经登录，如没有登录，则不进行交易，提示其进行登录。

在登录系统的部分，该系统通过登录验证去判断登录用户身份，从而进入各自的页面。页面部分还会显示出当前用户，而退出系统则是直接返回首页，将当前保存的登录用户信息清除。

登录系统的界面如图 15-15 所示。

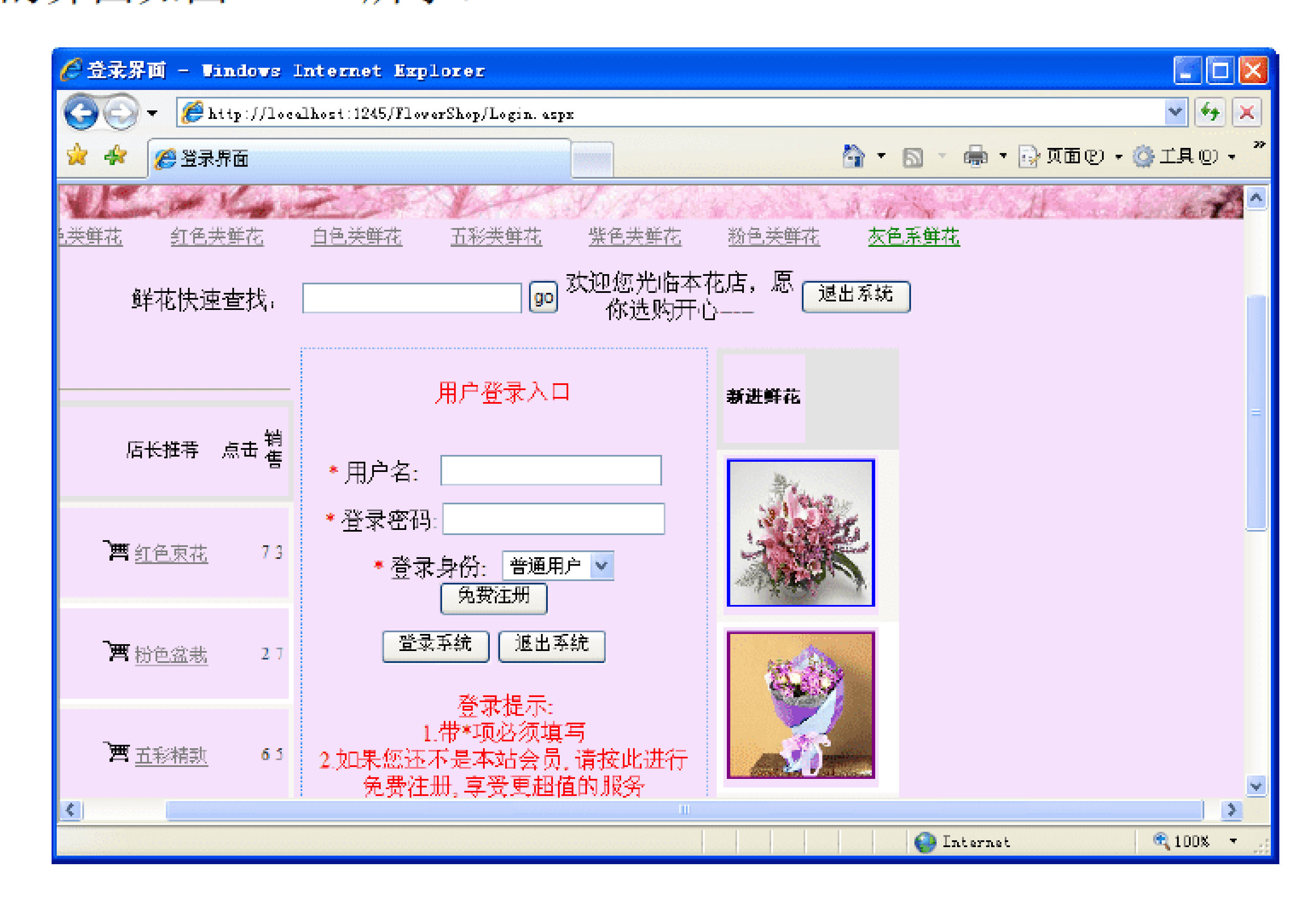


图 15-15　登录界面

成功登录后的界面如图 15-16 所示。



图 15-16　登录成功界面

15.5　管理员界面：用户管理

订购鲜花的会员在变化着，订单中又会涉及用户，所以管理好用户就相当于管理好了整个系统，本系统管理员拥有进入系统“后台管理”的权限，他可以查看本站注册会员的用户信息，可以对用户进行删除、修改权限的操作。

在管理员登录成功后，可以单击用户管理单元格内的用户图标，进入相应的用户管理页面，其设计界面如图 15-17 所示。



图 15-17　用户管理设计界面

其前台设计代码如下：

```
<tr>
   <td align="middle" style="width: 148px">
   <img onmouseover='this.style.cursor="hand"'
     onclick='OpenWin("Admin_User.aspx", "Member", 500, 250)'
     src="image/Forum.gif" border="0" style="height: 93px; width: 118px">
   </td>
   <td valign="top">
       管理用户<br> <br>※ 您可以查看用户、删除用户、更改用户权限
   </td>
</tr>
```



上述代码中，为单元格内的图标添加了一个样式，在 onclick 事件中的 OpenWin 函数中为链接到的 JScript.js 文件，代码如下所示。在管理员界面中，将此文件拖入即可。

```
function OpenWin(url, winName, width, height)
{
   xposition=0;
   yposition=0;
   if ((parseInt(navigator.appVersion) >= 4))
   {
       xposition = (screen.width-width)/2;
       yposition = (screen.height-height)/2;
   }
   theproperty = "width=" + width + ","
       + "height=" + height + ","
       + "location=0,"            //是否显示地址栏，1 为显示
```

```
        + "menubar=0,"              //是否显示菜单栏
        + "resizable=0,"            //是否允许改变窗口大小，1 为允许
        + "scrollbars=0,"           //是否显示滚动条，1 为允许
        + "status=0,"               //是否显示状态栏，1 为显示
        + "titlebar=0,"             //是否先是标题栏，1 为显示
        + "toolbar=0,"              //是否显示工具栏
        + "hotkeys=0,"
        + "screenx=" + xposition + "," //仅适用于 Netscape
        + "screeny=" + yposition + "," //仅适用于 Netscape
        + "left=" + xposition + ","  //IE
        + "top=" + yposition;     //IE
    monwin = window.open(url, winName, theproperty);
    monwin.focus();
}
```

登录成功后进入用户管理的具体信息页面(Admin_User.aspx)，其设计视图如图 15-18 所示。

MasterPage.master* | Admin_User.aspx*

MasterPage.master

鲜花快速查找： go [Label1] 退出系统

ContentPlaceHolder5 (自定义) | asp:GridView#GridView1

	权限ID	用户名	用户ID	联系电话	email	地址
编辑 删除	0	abc	0	abc	abc	abc
编辑 删除	1	abc	1	abc	abc	abc
编辑 删除	2	abc	2	abc	abc	abc
编辑 删除	3	abc	3	abc	abc	abc
编辑 删除	4	abc	4	abc	abc	abc

SqlDataSource - SqlDataSource1

设计 | 拆分 | 源 | <asp:GridView#GridView1> <Columns> <asp:CommandField>

图 15-18　用户管理页面设计视图

此页面为内容页，在左侧的店长推荐部分为用户自定义控件，店长随机推荐鲜花控件(hotflower.ascx)设计视图如图 15-19 所示。



图 15-19　店长随机推荐鲜花控件设计视图

Web程序员成功之路

店长随机推荐鲜花控件的前台页面代码如下：

```
<asp:DataList ID="DataList2" runat="server" CellPadding="4"
  ForeColor="#333333" Width="215px"
  Font-Size="10pt">
    <FooterStyle BackColor="#5D7B9D" Font-Bold="True" ForeColor="White" />
    <AlternatingItemStyle BackColor="White" ForeColor="#284775" />
    <ItemStyle BackColor="#F7F6F3" ForeColor="#333333" />
    <SelectedItemStyle BackColor="#E2DED6" Font-Bold="True"
      ForeColor="#333333" />
    <HeaderStyle BackColor="#E3E3E3" ForeColor="Black" />
    <HeaderTemplate>
        <table>
            <tr>
                <td style="width: 150px">
                    店长推荐
                </td>
                <td style="width: 40px">
                    点击
                </td>
                <td>
                    销售
                </td>
            </tr>
        </table>
    </HeaderTemplate>
    <ItemTemplate>
        <table>
            <tr>
                <td style="width: 150px">
                    <img src="image/ShoppingCart.gif" />
                    <a title="点击订购此鲜花"
                      href='<%#"LookFlower.aspx?id="+Eval("flowername") %>'>
                        <%# Eval("flowername")%></a>
                </td>
                <td style="width: 40px">
                    <%# Eval("spare")%>
                </td>
                <td>
                    <%# Eval("sale")%>
                </td>
            </tr>
        </table>
    </ItemTemplate>
</asp:DataList>
```

在此控件的 Page_Load 事件中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
```

```
    if (!IsPostBack)
    {
        Getflower();
    }
}
```

其中的 Getflower()方法的代码如下：

```
public void Getflower()
{
    ST Flower sf = new ST Flower(); //实例化一个 ST Flower 对象
    string s = "select top 6 * from flower order by newid()";//选择 6 个鲜花
    this.DataList2.DataSource =
      sf.GetDataSet(s).Tables[0].DefaultView; //设置数据源
    this.DataList2.DataBind(); //进行绑定
}
```

上述代码中的调用方法(sf.GetDataSet)为调用 ST_Flower 公共类中的方法，它返回 DataSet 数据集给页面层的用户自定义控件，然后将获取到的数据源绑定到自定义控件。

对于此页面核心的用户管理，用户显示及编辑用户的信息使用了 GridView 数据控件，具体对用户的操作，均通过数据源绑定时设置，如图 15-20 所示。

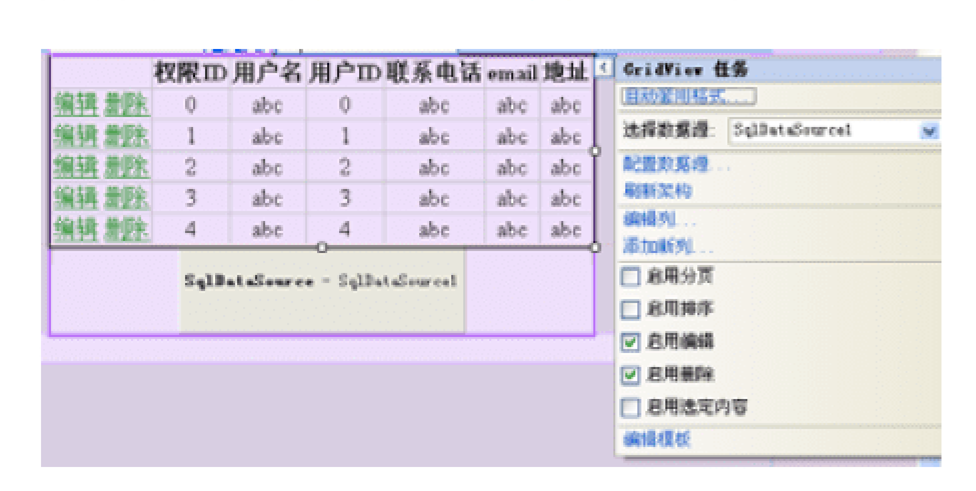


图 15-20　GridView数据源设计视图



利用 GridView 数据控件将数据库中的用户信息显示出来后，如需实现对用户的编辑和删除，则可直接选中此控件的“启用编辑”、“启用删除”功能选项。

用户管理页面的运行效果如图 15-21 所示。



图 15-21　用户管理页面的运行结果

15.6　管理员界面：鲜花管理

鲜花的购进卖出会让数据库中鲜花信息表持续地面临改变，固然少不了管理员对它的关联，本系统中在管理员登录后可以对鲜花进行查看、更改、删除的操作，也可以对新进的鲜花进行添加。

15.6.1　鲜花信息的查看和删除

本鲜花预订系统中，管理员可以对鲜花信息进行查看、更改和删除。

鲜花信息管理页面的设计视图如图 15-22 所示。



图 15-22　鲜花信息管理页面的设计视图

此页面的鲜花信息也是由数据控件 GridView 绑定，然后通过配置数据源，进行显示，再启用数据源的“删除”、“编辑”功能，其过程与用户管理的一致，在此不再赘述。

鲜花管理页面的运行结果如图 15-23 所示。

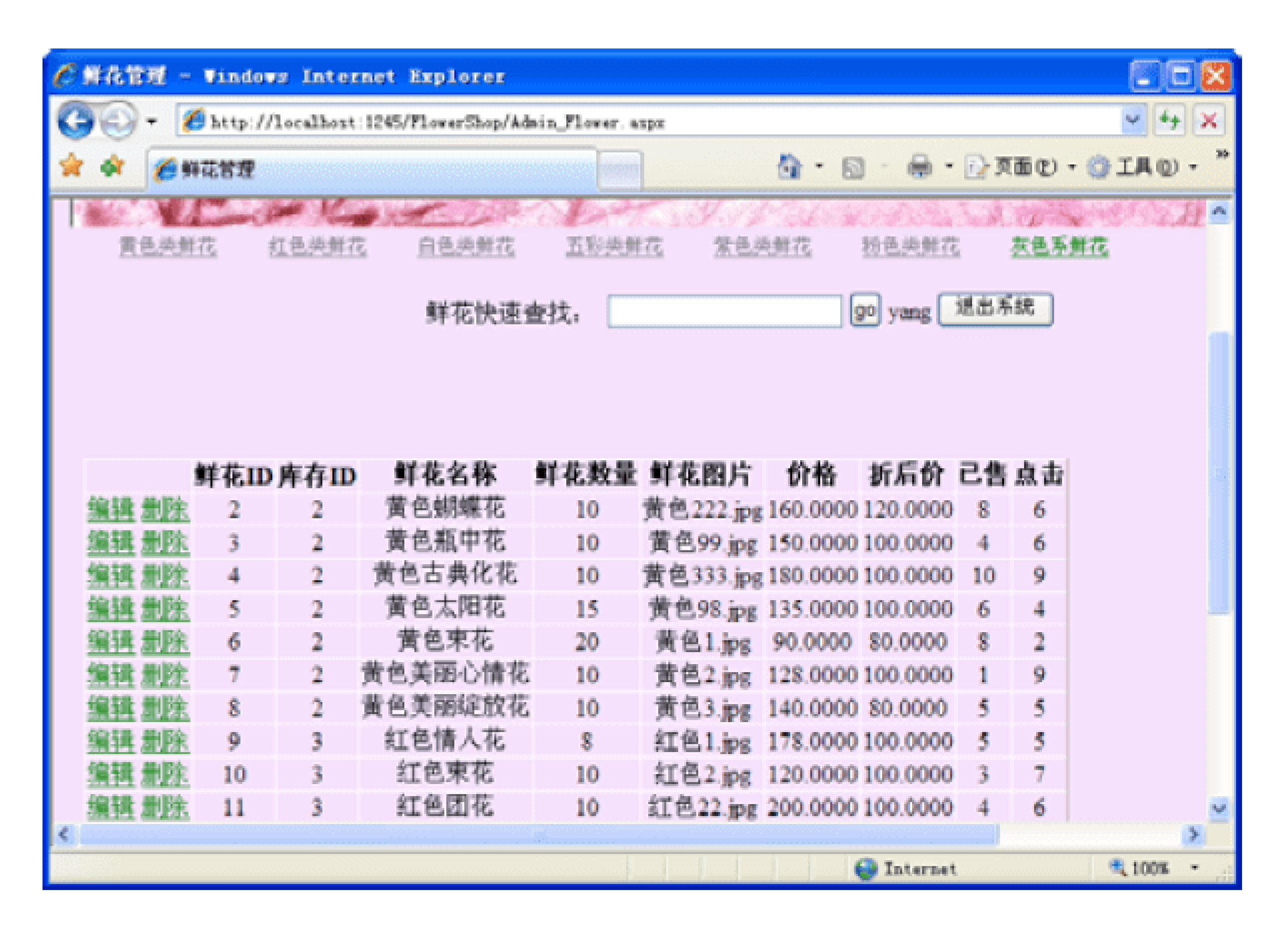


图 15-23　鲜花管理页面的运行结果

15.6.2 鲜花的添加

如果花店新进了一些不同品种的鲜花，管理员还可以实现对鲜花进行添加功能。

在管理员登录成功后，即进入管理员页面(Manager.aspx)。对于鲜花的添加，为方便快速，直接在页面中设计出来，其设计视图如图 15-24 所示。

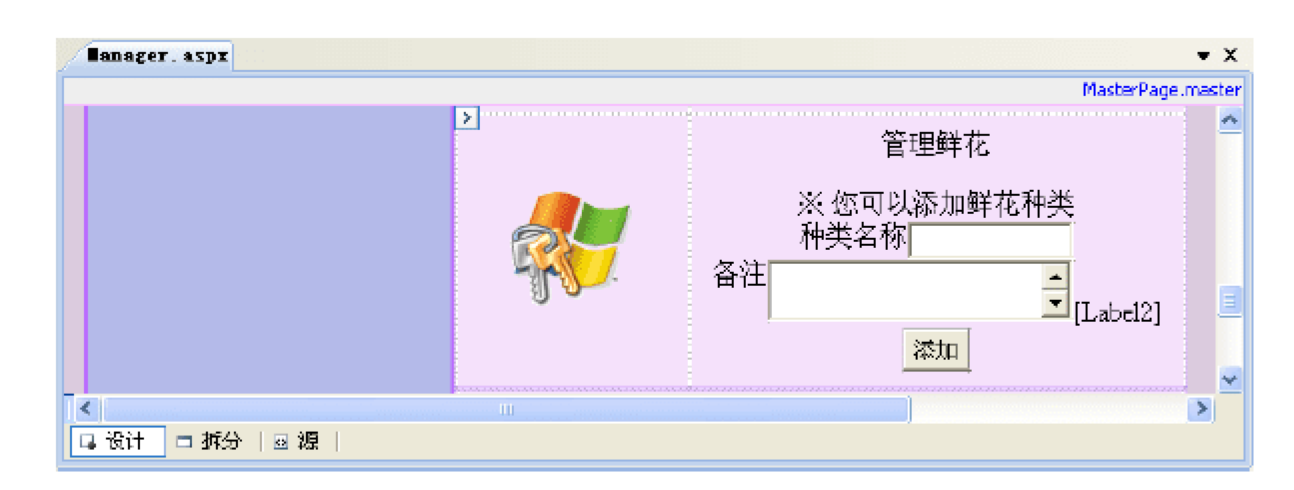


图 15-24 添加鲜花的设计视图

其前台页面部分代码如下：

```
<tr>
<td align="middle" style="width: 148px">
<img onmouseover='this.style.cursor="hand"'
  onclick='OpenWin("Admin_Flower.aspx", "Member", 500, 250)'
  src="image/Security.gif" border="0">
</td>
<td valign="top">
<font class="3DTitle">管理鲜花</font><br><br>
※ 您可以添加鲜花种类<br />
种类名称<asp:TextBox ID="txtcatename" runat="server" Columns="15"
  MaxLength="15" /><br />
备注<asp:TextBox ID="txtdemo" runat="server"
  TextMode="MultiLine"></asp:TextBox>
<asp:Label ID="Label2" runat="server"></asp:Label>
<asp:Button ID="Button1" runat="server" Text="添加" OnClick="Sumbit_Click" />
</td>
</tr>
```



上述代码中，为单元格内的图标添加上了一个样式，在 onclick 事件中的 OpenWin 函数中为链接到的 JScript.js 文件。

双击“添加”按钮，在其 Click 事件中添加如下代码：

```
protected void Sumbit_Click(object sender, EventArgs e)
{
    ST_Flower sf = new ST_Flower(); //实例化 St_Flower 类
    sf.Catename = txtcatename.Text; //将文本框的值赋给 sf 的成员
    sf.Demo = txtdemo.Text; //将文本框的值赋给 sf 的成员
    if (sf.CheckCateFlw(sf.Catename)) //判断是否存在该分类
    {
        sf.AddCateFlw(sf.Catename, sf.Demo); //将新的分类加入
```

```
        this.Label2.Text = "添加鲜花分类成功!"; //提示添加成功
    }
    else
    {
        this.Label2.Text = "此鲜花的种类已经存在!";
    }
}
```

上述代码中的有关 sf 调用的方法均来自 ST_Flower 公共类中，处理逻辑均在公共类中。添加前判断了鲜花是否存在的逻辑。

在鲜花种类中输入新进的新品种鲜花，可以加入备注信息，单击“添加”按钮，运行结果如图 15-25 所示。

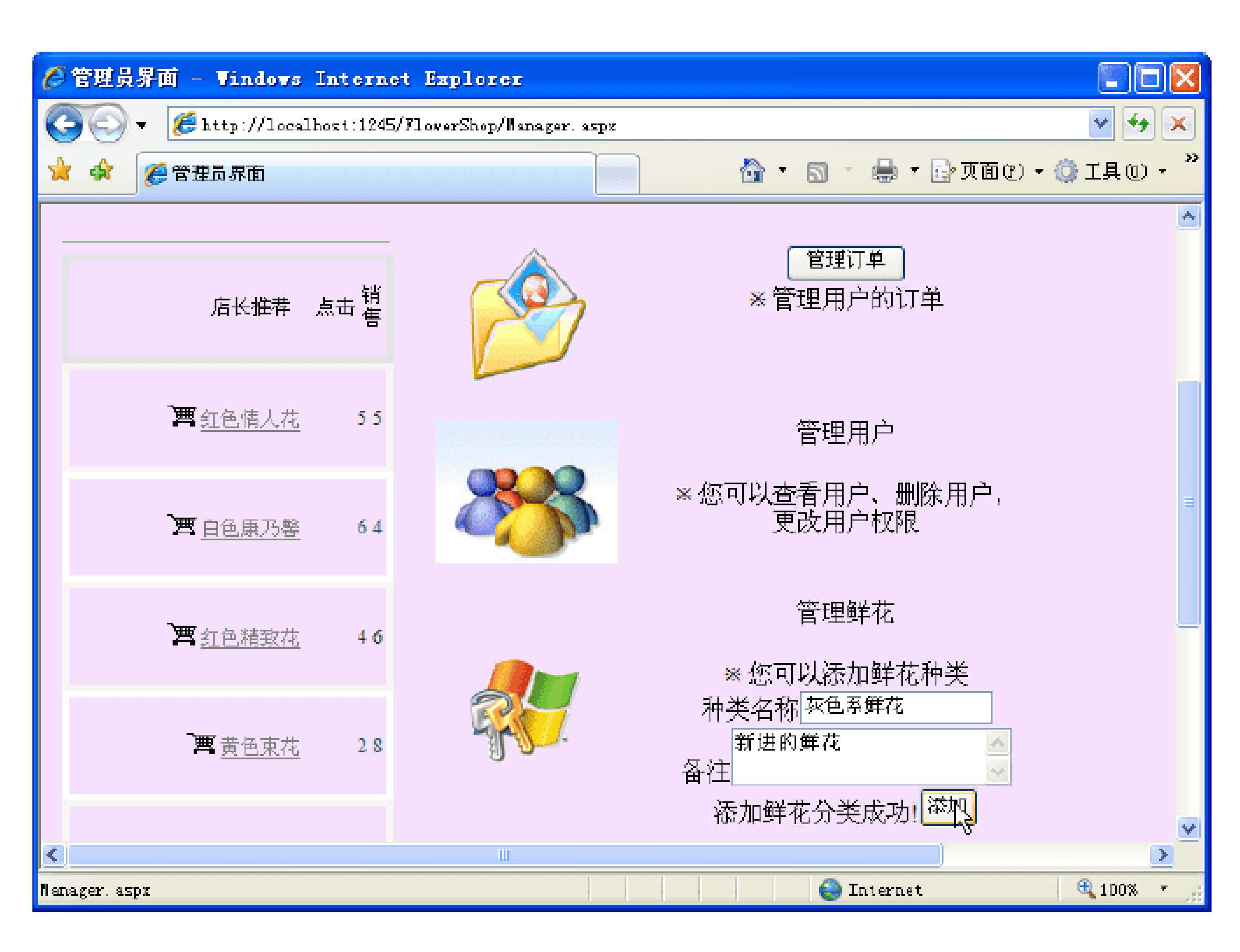


图 15-25　添加鲜花的运行效果

15.6.3　ST_Flower类

在上面章节的代码中，实现功能所用到的 ST_Flower 类起到了关键作用，本系统将所有与鲜花信息都有关的操作都封装在了这个类中，以便有整齐划一的感觉。

为与数据库中的表关联，此类封装了部分鲜花信息表中的字段，代码如下：

```
private double _total;
private string catename;
private string demo;
/// <summary>
/// 鲜花总价
/// </summary>
public double Total
{
    get { return _total; }
    set { _total = value; }
}
```

```
/// <summary>
/// 鲜花备注信息
/// </summary>
public string Demo
{
    get { return demo; }
    set { demo = value; }
}
/// <summary>
/// 库存名称
/// </summary>
public string Catename
{
    get { return catename; }
    set { catename = value; }
}
```

所有对鲜花信息的操作均离不开底层的数据库，因此操作中就少不了对数据库的连接，在此有必要将各个操作都需要用的连接数据库部分封装为一个函数。具体代码如下：

```
/// <summary>
/// 公用的打开 SqlConnection 连接
/// </summary>
/// <returns></returns>
public SqlConnection OpenCon()
{
    string str = ConfigurationManager.ConnectionStrings["constr"]
      .ConnectionString; //获取连接字符串
    SqlConnection con = new SqlConnection(str);
    con.Open(); //打开连接
    return con; //返回一个 SqlConnection
}
```



在读取连接字符串时，均为从配置文件 Web.config 中读取。对数据库的操作需引入 System.Data 和 System.Data.SqlClient 命名空间。

在管理员对鲜花进行添加时，需要验证要添加的鲜花是否在库存中已经存在，在此类中由 CheckCateFlw 函数进行验证，验证通过才可以进行添加，具体实现代码如下：

```
/// <summary>
/// 查看库存中是否有重复的鲜花类别
/// </summary>
/// <param name="catename"></param>
/// <returns></returns>
public bool CheckCateFlw(string catename)
{
    SqlCommand cmd = new SqlCommand("selclass", OpenCon());
    cmd.CommandType = CommandType.StoredProcedure; //命令类型为存储过程
    SqlParameter sp1 = cmd.Parameters.Add("@catename", SqlDbType.VarChar);
    sp1.Value = catename;
```

```
    SqlDataReader sdr = cmd.ExecuteReader();
    if (sdr.Read())
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

经过上述函数的验证通过后，才可以真正地将新进鲜花种类添加到数据库中，下面的方法执行了此添加过程，具体代码如下：

```
/// <summary>
/// 添加此鲜花分类
/// </summary>
/// <param name="flwclassnm"></param>
/// <param name="flwdemo"></param>
public void AddCateFlw(string flwclassnm, string flwdemo)
{
    SqlCommand cmd = new SqlCommand("addclass", OpenCon());
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameter classname =
      cmd.Parameters.Add("@catename", SqlDbType.VarChar); //加入参数
    classname.Value = flwclassnm; //给参数赋值
    SqlParameter demo = cmd.Parameters.Add("@demo", SqlDbType.NVarChar);
    demo.Value = flwdemo;
    cmd.ExecuteNonQuery(); //执行插入语句
    OpenCon().Close(); //关闭连接
}
```

无论是游客、会员还是管理员都可以对鲜花进行查看和搜索，实现此功能的函数具体代码如下：

```
/// <summary>
/// 获得鲜花信息
/// </summary>
/// <param name="s"></param>
/// <returns></returns>
public DataSet Getgo(string s)
{
    SqlCommand com = new SqlCommand("p_select", OpenCon());
    com.CommandType = CommandType.StoredProcedure;
    SqlParameter sp = new SqlParameter("@flowername", DbType.String);
    sp.Value = s;
    com.Parameters.Add(sp);
    DataSet ds = new DataSet();
    new SqlDataAdapter(com).Fill(ds);
    return ds; //将得到的数据作为 DataSet 返回调用方
```

```
}
```

在会员对鲜花进行订购时，将产生当前用户所挑选的鲜花具体信息，如鲜花名称、鲜花数量、单价及所需支付的总价。具体实现代码如下：

```
/// <summary>
/// 获得当前所订购的鲜花信息
/// </summary>
/// <param name="a"></param>
/// <param name="s"></param>
/// <returns></returns>
public DataTable GetChat(ArrayList a, ArrayList s)
{
    DataTable dt = new DataTable("gouwu"); //实例化 DataTable
    DataRow dr1 = dt.NewRow(); //创建新的行
    DataColumn dc1 = new DataColumn(); //创建新的列
    dc1.DataType = typeof(string); //设置列的数据类型
    dc1.ColumnName = "flowername"; //设置列名
    dt.Columns.Add(dc1); //将列加入到表中
    DataColumn dc2 = new DataColumn();
    dc2.DataType = typeof(int);
    dc2.ColumnName = "num";
    dt.Columns.Add(dc2);
    DataColumn dc3 = new DataColumn();
    dc3.DataType = typeof(string);
    dc3.ColumnName = "disprice";
    dt.Columns.Add(dc3);
    DataColumn dc4 = new DataColumn();
    dc4.DataType = typeof(int);
    dc4.ColumnName = "total";
    dt.Columns.Add(dc4);
    for (int i=0; i<a.Count; i++)
    {
        string b = a[i].ToString();
        dr1 = GetRow(b);
        DataRow dr = dt.NewRow();
        dr["flowername"] = dr1["flowername"];
        dr["num"] = s[i];
        dr["disprice"] = dr1["disprice"];
        dr["total"] =
          double.Parse(dr1["disprice"].ToString()) * ((int)dr["num"]);
        this._total +=
          double.Parse(dr1["disprice"].ToString()) * ((int)dr["num"]);
        dt.Rows.Add(dr);
    }
    return dt;
}
/// <summary>
/// 返回一行
```

```
/// </summary>
/// <param name="a"></param>
/// <returns></returns>
public DataRow GetRow(string a)
{
    string s = "select flowername,disprice from flower where flowername='"
                + a + "'";
    return GetDataSet(s).Tables[0].Rows[0];
}
/// <summary>
/// 返回 DataSet 数据集
/// </summary>
/// <param name="sql"></param>
/// <returns></returns>
public DataSet GetDataSet(string sql)
{
    DataSet ds = new DataSet();
    new SqlDataAdapter(sql, OpenCon()).Fill(ds);
    return ds;
}
```

在用户确认了购买以上所显示的鲜花订购信息后，生成订单则需要将当前订购信息插入数据库中的订单信息表中。具体实现代码如下：

```
/// <summary>
/// 将所订购鲜花加入订单
/// </summary>
/// <param name="a"></param>
/// <param name="b"></param>
/// <param name="username"></param>
public void AddToCart(ArrayList a, ArrayList b, string username)
{

    DataTable dt = GetChat(a, b);
    foreach (DataRow datarow in dt.Rows)
    {
        string sql = @"insert into shoppingcart
                (flowername,flwnum,price,total,username)"
              + "values('" + datarow["flowername"].ToString() + "'"
              + ",'" + datarow["num"].ToString() + "'"
              + ",'" + datarow["disprice"].ToString() + "'"
              + ",'" + datarow["total"].ToString() + "'"
              + ",'" + username + "')"; //向订单表中插入的 SQL 语句
        SqlCommand sqlcommand = new SqlCommand(sql, OpenCon());
        sqlcommand.ExecuteNonQuery(); //执行插入语句
        OpenCon().Close(); //关闭连接
    }
}
```

以上 ST_User 类中的方法大都返回 DataSet 数据集，然后在调用方进行绑定，无非根据不

同的需求，更改相应的 SQL 查询语句而已。SqlCommand 命令类型设置为存储过程的均为调用的存储过程。

至此，ST_Flower 公共类构建完毕。



对涉及到鲜花的操作函数都放入此类，大大简化了操作，在页面部分使用时，只需将此类实例化再调用相应的函数即可。

15.7 管理员界面：订单管理和信息查找

会员用户进行了鲜花订购，管理员就要查看哪些用户订购了哪些鲜花，以便能及时处理订单，将顾客的鲜花送达目的地，本鲜花预订系统的管理员可以对因用户订购所产生的订单进行操作，包括删除、查看订单。

15.7.1 订单信息的查看和处理

在系统管理登录成功进入管理员界面时，可以查看到订单的详细信息，如果是已经发货的则可以将此订单删除，其设计视图如图 15-26 所示。

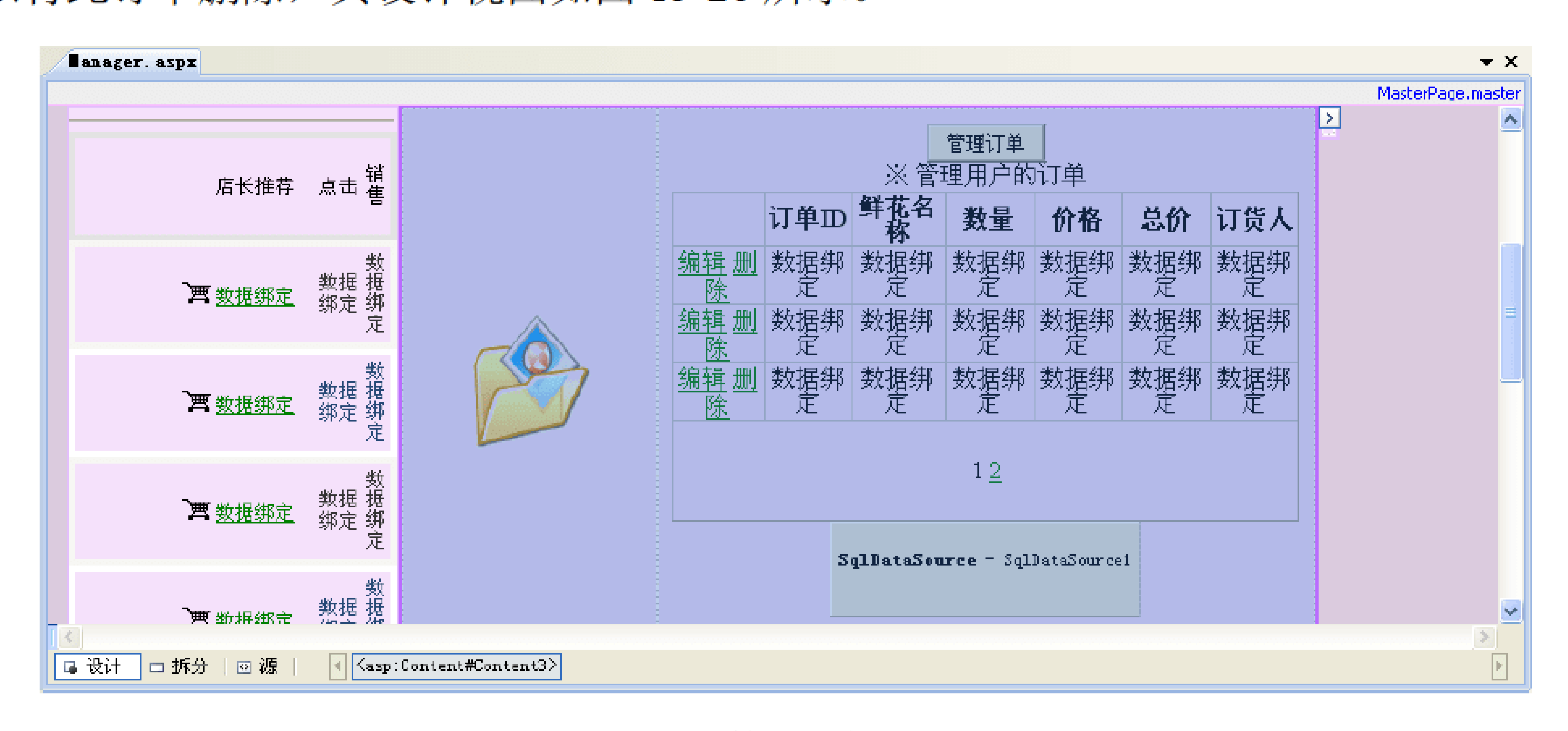


图 15-26 管理订单设计视图

在上面的设计部分用到了 GridView 数据控件，并且配置了数据源，启用了“删除”与“编辑”功能选项，这部分的具体实现与 15.5 节的用户管理大体一致，此处不再赘述。

其中的“管理订单”按钮为显示订单所用。

在页面的 Page_Load 事件中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    this.GridView2.Visible = false;
}
```

在“管理订单”按钮的 Click 事件中添加如下代码：

```
protected void Button2_Click(object sender, EventArgs e)
{
    this.GridView2.Visible = true;
}
```

单击“管理订单”按钮，运行结果如图 15-27 所示。

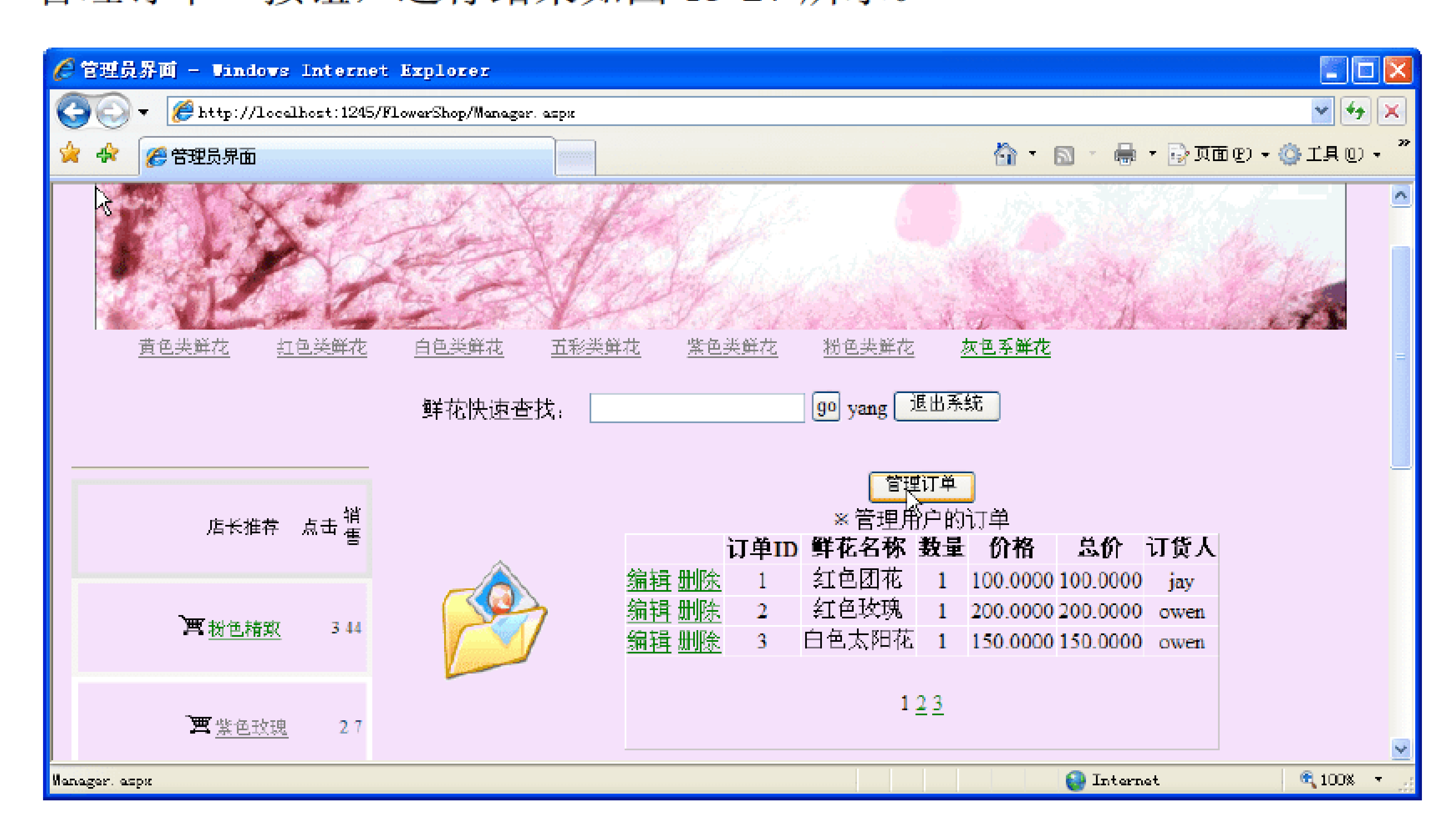


图 15-27　管理订单运行结果

15.7.2　ST_User类

同 ST_Flower 类一样，对于用户的所有操作均放在 ST_User 类中，此类封装了所有有关用户的操作处理函数，在需要使用时，只需实例化一个本类的对象，然后调用相应的方法。

首先，将数据库中用户信息表的部分字段封装起来，具体代码如下所示：

```
private string username;
private string password;
private int limitid;
/// <summary>
/// 用户名
/// </summary>
public string Username
{
    get { return username; }
    set { username = value; }
}
/// <summary>
/// 密码
/// </summary>
public string Password
{
    get { return password; }
    set { password = value; }
```

```
}
/// <summary>
/// 标示用户身份 ID
/// </summary>
public int Limitid
{
    get { return limitid; }
    set { limitid = value; }
}
```

在用户登录本网站时，需要验证此用户是何种身份，从而显示不同的页面，验证部分的具体代码如下：

```
/// <summary>
/// 验证用户身份
/// </summary>
/// <param name="username"></param>
/// <param name="password"></param>
/// <param name="limitid"></param>
/// <returns></returns>
public SqlDataReader CheckUser(string username, string password, int limitid)
{
    SqlCommand cmd = new SqlCommand("checkuser",  OpenCon());
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameter usernm = cmd.Parameters.Add("@username", SqlDbType.NVarChar);
    usernm.Value = username;
    SqlParameter pwd = cmd.Parameters.Add("@password",SqlDbType.NVarChar);
    pwd.Value = password;
    SqlParameter limid = cmd.Parameters.Add("@limitid",SqlDbType.Int);
    limid.Value = limitid;
    SqlDataReader sdr = cmd.ExecuteReader();
    return sdr;
}
```

本系统为会员提供了修改密码的功能，具体实现代码如下：

```
/// <summary>
/// 用户更改密码
/// </summary>
/// <param name="name"></param>
/// <param name="str1"></param>
/// <param name="str2"></param>
public void spCustomersUpdatePassword(string name, string str1, string str2)
{
    SqlCommand cmd = new SqlCommand("updatepwd", OpenCon());
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameter usernm = cmd.Parameters.Add("@username", SqlDbType.NVarChar);
    usernm.Value = name;
    SqlParameter oldpwd = cmd.Parameters.Add("@oldpwd", SqlDbType.NVarChar);
    oldpwd.Value = str1;
```

```
    SqlParameter newpwd = cmd.Parameters.Add("@newpwd", SqlDbType.NVarChar);
    newpwd.Value = str2;
    cmd.ExecuteNonQuery();
}
```

在登录过程还要验证登录用户是否存在，具体代码如下：

```
/// <summary>
/// 验证用户名是否存在
/// </summary>
/// <param name="name"></param>
/// <returns></returns>
public bool CheckUser(string name)
{
    string sql =
      "select * from users where username='" + name + "'"; //查询语句
    SqlCommand cmd = new SqlCommand(sql, OpenCon()); //设置命令对象
    SqlDataReader sdr = cmd.ExecuteReader();
    if (sdr.Read()) //判断数据库有无此用户
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

对于不存在的用户，系统提供注册功能，具体代码如下：

```
/// <summary>
/// 注册新用户
/// </summary>
/// <param name="name"></param>
/// <param name="pwd"></param>
/// <param name="email"></param>
/// <param name="phone"></param>
/// <param name="address"></param>
public void AddUser(string name, string pwd, string email, string phone,
  string address)
{
    SqlCommand cmd = new SqlCommand("adduser", OpenCon());
    cmd.CommandType = CommandType.StoredProcedure;
    SqlParameter username =
      cmd.Parameters.Add("@username", SqlDbType.NVarChar);
    username.Value = name;
    SqlParameter password =
      cmd.Parameters.Add("@password", SqlDbType.NVarChar);
    password.Value = pwd;
    SqlParameter useremail = cmd.Parameters.Add("@email", SqlDbType.NVarChar);
```

```
    useremail.Value = email;
    SqlParameter userphone =
      cmd.Parameters.Add("@phonenum", SqlDbType.NVarChar);
    userphone.Value = phone;
    SqlParameter useraddress =
      cmd.Parameters.Add("@address", SqlDbType.NVarChar);
    useraddress.Value = address;
    cmd.ExecuteNonQuery();
}
```

用户可以对自己所订购的鲜花信息订单进行查看，具体实现代码如下：

```
/// <summary>
/// 查找个人订单
/// </summary>
/// <param name="username"></param>
/// <returns></returns>
public DataSet SearchMyOrder(string username)
{
    string sql = "select * from shoppingcart where username='"
                   + username + "'"; //查看个人订单语句

    SqlCommand cmd = new SqlCommand(sql, OpenCon());
    SqlDataAdapter sda = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
    sda.Fill(ds);
    return ds;
}
```

管理员则可以查看所有用户的订单信息，以实现对订单的操作，具体实现代码如下：

```
/// <summary>
/// 查看所有订单
/// </summary>
/// <returns></returns>
public DataSet SearchOrders()
{
    string sql = "select * from shoppingcart "; //从购物车表得出订单信息
    SqlCommand cmd = new SqlCommand(sql, OpenCon());
    SqlDataAdapter sda = new SqlDataAdapter(cmd);
    DataSet ds = new DataSet();
    sda.Fill(ds);
    return ds;
}
```

以上 ST_User 类中的方法大都返回 DataSet 数据集，然后在调用方进行绑定，无非根据不同的需求，更改相应的 SQL 查询语句而已。SqlCommand 命令类型为存储过程的均为调用的存储过程。

至此，所有与用户操作相关的功能函数都封装在 ST_User 公共类中。

15.7.3 信息查找

本系统提供对鲜花信息的查找，将此功能添加到了头部控件(header.ascx)中，其代码部分在 15.4.3 小节已经给出，此部分涉及到另一个用户自定义控件(chakan.ascx)，用于显示所查询鲜花的信息。此控件由 DataList 数据控件组成，其设计视图如图 15-28 所示。

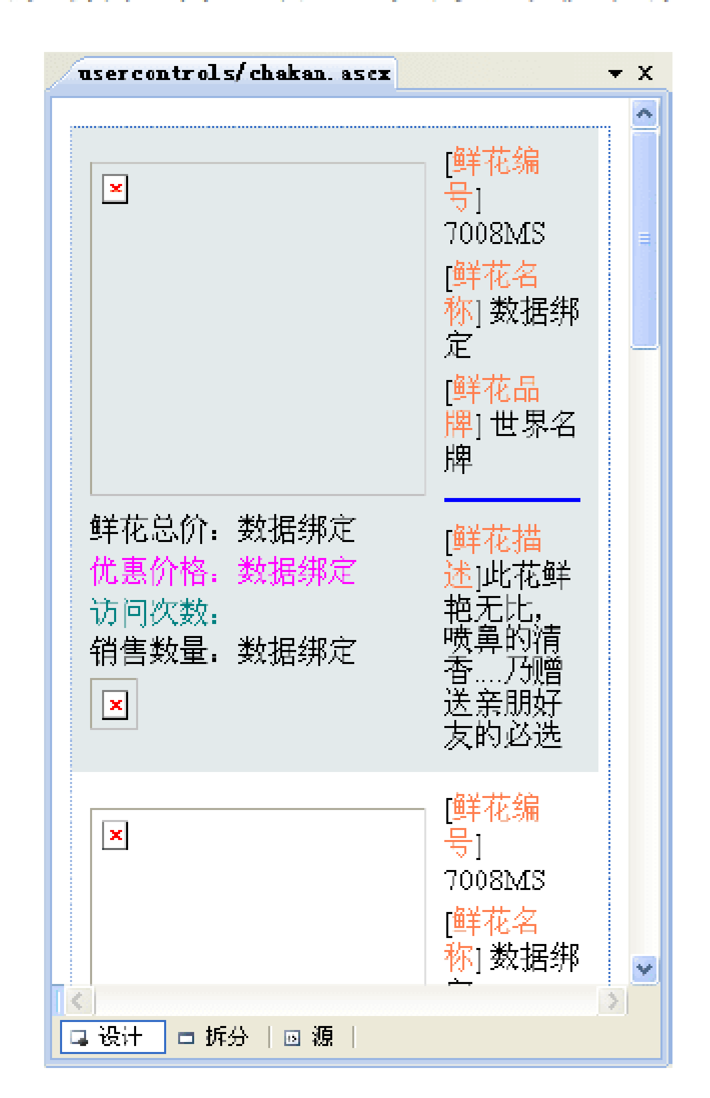


图 15-28 查看鲜花控件设计视图

实现此用户自定义控件的后台代码如下：

```
protected void Page_Load(object sender, EventArgs e)
{
    Getchakan(); //调用查看鲜花信息的函数
    if (!IsPostBack)
    {
        Getcishu(); //调用查看次数的函数
    }
}
/// <summary>
/// 显示查看的鲜花信息
/// </summary>
public void Getchakan()
{
    string s = Request["id"]; //取得当前所选择的鲜花 ID 号
    this.DataList1.DataSource =
      sf.Getgo(s).Tables[0].DefaultView; //获得当前要查看鲜花的信息
    this.DataList1.DataBind(); //进行绑定
}
/// <summary>
/// 获得查看次数
/// </summary>
```

```
public void Getcishu()
{
    ArrayList flowerid = (ArrayList)Application["flowerid"];
    ArrayList num = (ArrayList)Application["num"];
    if (flowerid.IndexOf(Request["id"]) == -1)
    {
        flowerid.Add(Request["id"]);
        num.Add("1");
    }
    else
    {
        int a = flowerid.IndexOf(Request["id"]);
        num[a] = Convert.ToInt32(num[a]) + 1;
    }
    for (int i=0; i<this.DataList1.Items.Count; i++)
    {
        Label lb = (Label)DataList1.Items[i].FindControl("Label1");
        lb.Text = num[flowerid.IndexOf(Request["id"])].ToString();
    }
}
```

对于用户的查看鲜花的具体信息，都是用 DataList 数据控件进行绑定，关于查找鲜花的信息，上述代码只需传入 id，然后根据相应查询语句即可得到此鲜花的信息。

15.8 一般用户界面

对于游客来说，是不能订购鲜花的，只有验证通过的普通会员用户可以进行鲜花订购以及查看自己的订单信息、修改密码与查找鲜花。本系统给一般用户提供了这些功能。

15.8.1 购物车

购物车毫无疑问是每个网上购物系统的核心，本网站的购物车为用户控件(chat.ascx)，它的主体部分是由 GridView 数据控件组成，其设计视图如图 15-29 所示。



图 15-29 购物车控件的设计视图

在 Page_Load 事件中添加如下代码：

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        if (Session["username"].ToString() != "") //判断当前用户是否为游客身份
        {
            GetChat();
            GetBind(numid, b);
        }
        else
        {
            this.Label2.Text = "你还没有订购任何一种鲜花！";
        }
    }
}
```

双击“全部选中”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 将所有 CheckBox 都设为选中状态
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Button1_Click(object sender, EventArgs e)
{
    for (int i=0; i<GridView1.Rows.Count; i++) //对 GridView 中的行进行循环遍历
    {
        CheckBox cb = (CheckBox)this.GridView1.Rows[i]
          .FindControl("CheckBox1"); //查找 CheckBox 类型的控件
        cb.Checked = true; //将 CheckBox 设为选中状态
    }
}
```

双击“全部取消”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 将所有 CheckBox 都设为取消状态
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Button2_Click(object sender, EventArgs e)
{
    for (int i=0; i<GridView1.Rows.Count; i++)
    {
        CheckBox cb = (CheckBox)this.GridView1.Rows[i]
          .FindControl("CheckBox1"); //查找 CheckBox 类型的控件
        cb.Checked = false; //将 CheckBox 设为未选状态
    }
}
```

上述代码中在对 CheckBox 的选中状态进行更改时，用到 FindControl 方法先行查找出其中的控件，然后再设置它的状态。

双击“全部删除”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 删除所选订购鲜花
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Button3_Click(object sender, EventArgs e)
{
    for (int i=0; i<GridView1.Rows.Count; i++)
    {
        CheckBox cb = (CheckBox)GridView1.Rows[i].FindControl("CheckBox1");
        if (cb.Checked)
        {
            numid = (ArrayList)Session["flowerid"];
            b = (ArrayList)Session["num"];
            numid.Remove(numid[i]);
            b.Remove(b[i]); //将选中订购的鲜花移除
            GetBind(numid, b); //重新绑定
        }
    }
}
```

双击“更新数量”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 更新订购数量
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    numid = (ArrayList)Session["flowerid"];
    b = (ArrayList)Session["num"];
    for (int i=0; i<GridView1.Rows.Count; i++)
    {
        TextBox tb = (TextBox)this.GridView1.Rows[i].FindControl("TextBox1");
        CheckBox cb =
          (CheckBox)this.GridView1.Rows[i].FindControl("CheckBox1");
        if (cb.Checked)
        {
            b[i] = tb.Text;
        }
    }
    GetBind(numid, b);
}
```

在对购物车中的鲜花订购信息进行更新删除等操作时，都需要判断是否选中，即上述代码中的 cb.Checkd，进行相应的操作后，就要重新将更新的信息绑定到购物车上。

双击“结账付款”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 结账付款
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void ImageButton2_Click(object sender, ImageClickEventArgs e)
{
    if (GridView1.Rows.Count > 0) //判断购物车是否为空
    {
        Response.Redirect("shoppingchat.aspx"); //跳转到购物车页面
    }
    else
    {
        Label2.Text = "系统提示：您当前的购物清单是空的，所以，您还无需进行结账付款.";
    }
}
```

在跳转到购物车页面时，需判断购物车内有无订购的信息才可以跳转。

在以上代码中均用到了 GetBind()与 GetChat()方法，这两个方法将购物车信息显示在用户面前。具体实现代码如下：

```
public void GetChat()
{
    numid = (ArrayList)Session["flowerid"];
    b = (ArrayList)Session["num"];
    if (numid.IndexOf(Request["id"]) == -1)
    {
        numid.Add(Request["id"]);
        b.Add("1");
    }
    else
    {
        int num = numid.IndexOf(Request["id"]);
        b[num] = Convert.ToInt32(b[num]) + 1;
    }
}
public void GetBind(ArrayList numid, ArrayList b)
{
    numid = (ArrayList)Session["flowerid"];
    b = (ArrayList)Session["num"];
    this.GridView1.DataSource = sf.GetChat(numid, b);
    this.GridView1.DataBind();
    this.Label1.Text = sf.Total.ToString() + "元";
}
```

一些页面都已经设置好了，下面就来进行一次购物模拟，看看具体的实现流程吧。首先在用户已经登录成功后，进行鲜花订购，运行结果如图 15-30 所示。



图 15-30　用户查看鲜花以进行订购

单击“订购此商品”按钮，运行结果如图 15-31 所示。



图 15-31　订购鲜花运行结果

将文本框中的数量更改为 3，再单击“更新数量”按钮，运行结果如图 15-32 所示。

如果确认了订购这些鲜花，单击“结账付款”按钮，运行结果如图 15-33 所示。

若单击“删除购物”按钮，运行结果如图 15-34 所示。



图 15-32　单击“更新数量”按钮的运行结果



图 15-33　单击“结账付款”按钮的运行结果



图 15-34　单击“删除购物”按钮的运行结果

15.8.2 用户注册

在以游客身份进行鲜花订购时，提供了用户注册功能，在用户登录界面，单击“免费注册”按钮，将跳转到 UserRegister.aspx 页面，页面中运用了 RegularExpressionValidator 验证控件来验证用户输入的电子邮件格式，此页面的设计视图如图 15-35 所示。

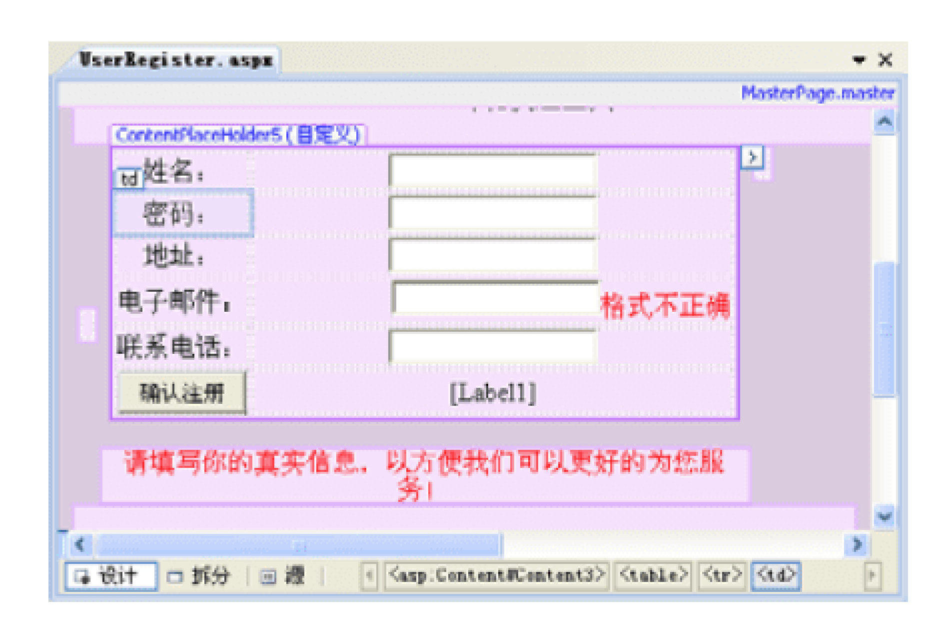


图 15-35 注册页面的设计视图

双击“确认注册”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 注册新用户
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void Button1_Click(object sender, EventArgs e)
{
    ST_User user = new ST_User();
    string name = txtusename.Text;
    string pwd = txtpwd.Text;
    string email = txtemail.Text;
    string phone = txtphone.Text;
    string address = txtaddress.Text;
    if (!user.CheckUser(name)) //判断用户名是否存在
    {
        this.Label1.Text = "此用户名已经存在，请更换一个用户名！";
        txtusename.Text = "";
    }
    else
    {
        user.AddUser(name,pwd,email,
          phone,address); //将此注册用户加入数据库 users 表
        Label1.Text = "注册成功！";
    }
}
```

上述代码在执行将新注册用户添加到数据库之前，需要判断此注册用户名是否存在，即 user.Check 方法。为方便输入重复用户名的用户反复输入信息，在提示用户名存在后，只将输

入用户名的文本框清空。



其中调用的 user.AddUser 方法在 15.7.2 小节中已经详细给出。

在跳转到用户注册页面后，输入一些注册的用户信息，单击“确认注册”按钮，完成新用户的注册。运行结果如图 15-36 所示。

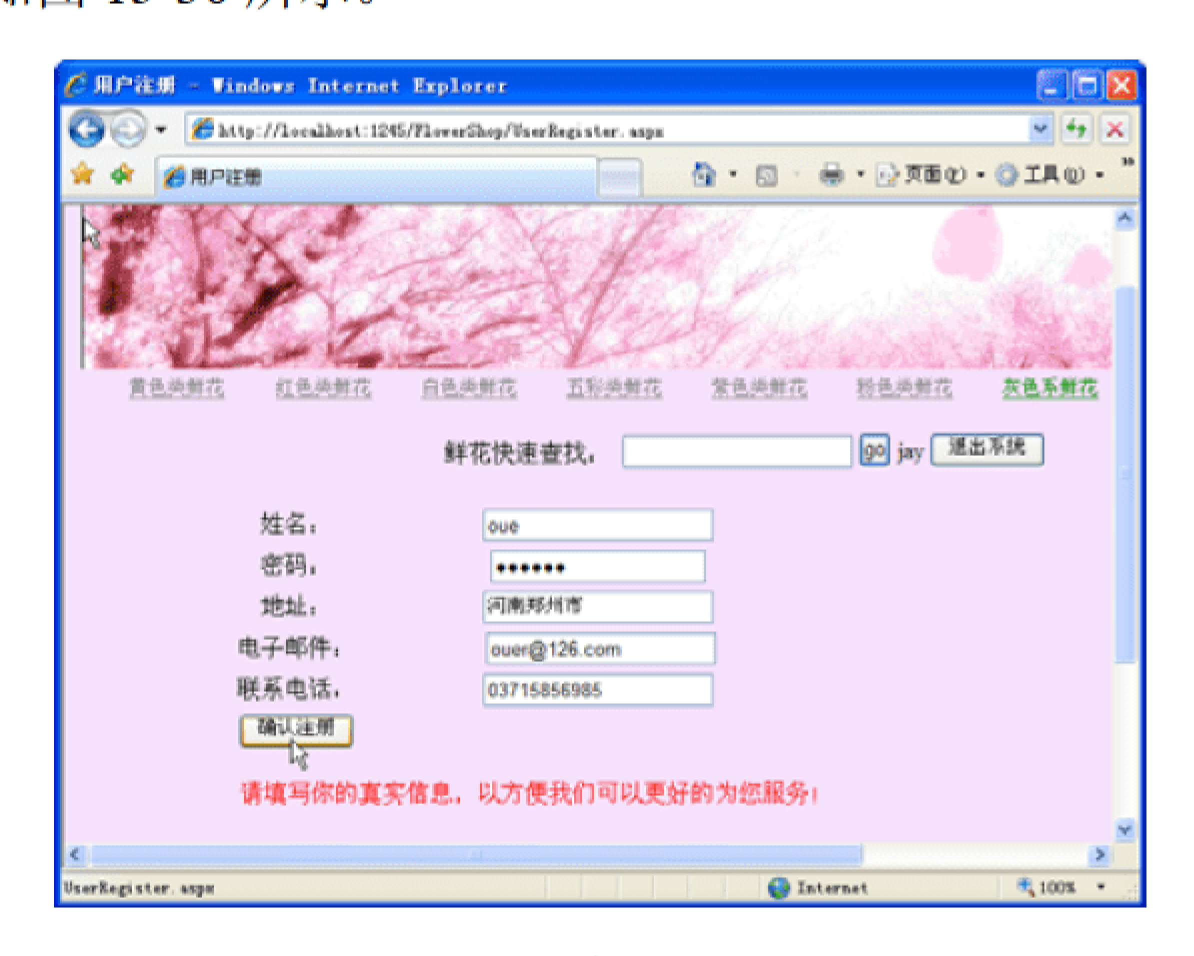


图 15-36　用户注册运行结果

15.8.3　我的订单

在用户已经完成鲜花订购后，可以查看自己所订购的鲜花，在用户登录成功后的页面单击“查看我的订单”按钮，可以查看自己的订单。“我的订单”设计视图如图 15-37 所示。



图 15-37　“我的订单”设计视图

双击“查看我的订单”按钮，进入其 Click 事件，添加如下代码：

```
/// <summary>
/// 查看个人订单
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
```

```
protected void Button2_Click(object sender, EventArgs e)
{
    string username = Session["username"].ToString();
    ST_User st = new ST_User();
    DataSet ds = st.SearchMyOrder(username); //查找个人订单，返回 DataSet 数据集
    this.GridView1.DataSource = ds; //将返回数据集作为数据源
    this.GridView1.DataBind(); //绑定数据源
}
```

上述代码实现了查找个人订单。怎样避免将数据库中的所有订单查询出来呢？在此将当前登录的用户名作为一个查询参数传入查询语句，即可从数据库中筛选出自己的订单信息。



其中调用的 st.SearchMyOrder 方法在 15.7.2 小节中已经详细给出。其中传入的 username 参数，通过页面储存的 Session 变量取得，从而取得当前用户的订单详细信息。

在运行中，单击“查看我的订单”按钮，运行结果如图 15-38 所示。



图 15-38　单击“查看我的订单”按钮的运行结果

15.8.4　用户密码修改

经常修改密码可以有效地保护自己账户的安全性，本系统网站为会员用户提供了修改密码的功能。修改密码的设计视图如图 15-39 所示。



图 15-39　修改密码设计视图

运行时单击图片即可进入密码修改页面(Security.aspx)，页面应用了 RequiredFieldValidator 和 CompareValidator 验证控件，用于提示用户输入不能为空及两次密码输入需一致。其设计视图如图 15-40 所示。



图 15-40　密码修改页面的设计视图

密码修改页面的前台代码如下：

```
<asp:Panel ID="Panel1" runat="server" BorderColor="Gray"
  BorderStyle="Dotted" BorderWidth="1px" Width="100%">
    <table border="0" cellpadding="10" cellspacing="0" width="100%">
        <tr>
            <td align="middle">
                <img border="0" src="image/Security.gif"/>
                 
            </td>
            <td valign="top">
                <font class="3DTitle">修改密码</font> <br>
                ※ 为安全起见，请不定期更改您的登录密码
                <br>
                <asp:Label ID="Label1" runat="server" ForeColor="Red">
                </asp:Label>
            </td>
        </tr>
    </table>
    <table border="0" cellpadding="5" cellspacing="0">
        <tr>
            <td width="55%">
                旧的密码：
                <asp:TextBox ID="TextBox1" runat="server"
                  BorderWidth="1px" TextMode="Password">
                </asp:TextBox>
            </td>
            <td>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
                  runat="server" ControlToValidate="TextBox1"
                  Display="Dynamic" ErrorMessage="修改密码前必须输入旧密码">
                </asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td>
                 新的密码：
                <asp:TextBox ID="TextBox2" runat="server"
```

```
                BorderWidth="1px" TextMode="Password">
              </asp:TextBox>
          </td>
          <td>
              <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
                runat="server" ControlToValidate="TextBox2"
                Display="Dynamic" ErrorMessage="请输入新的密码">
              </asp:RequiredFieldValidator>
          </td>
      </tr>
      <tr>
          <td>
              确认密码:
              <asp:TextBox ID="TextBox3" runat="server" BorderWidth="1px"
                TextMode="Password">
              </asp:TextBox>
          </td>
          <td>
              <asp:CompareValidator ID="CompareValidator1" runat="server"
                ControlToCompare="TextBox2" ControlToValidate="TextBox3"
                Display="Dynamic" ErrorMessage="两次输入的密码不一样">
              </asp:CompareValidator>
          </td>
      </tr>
      <tr>
          <td>
              <asp:Button ID="Button1" runat="server" BorderWidth="1px"
                OnClick="Button1_Click" Text="保存修改" />
          </td>
      </tr>
  </table>
</asp:Panel>
```

上述代码中，因修改的为密码，所以均将 TextBox 的 TextMode 属性设置为 Password。而且所用到的验证控件需要注意设置 ControlToCompare 的属性时，不要设置错了相应的文本框。

此页面运行结果如图 15-41 所示。



图 15-41　密码页面的运行结果

在各个相应文本框中输入密码，单击“保存修改”按钮，运行结果依次如图 15-42 和图 15-43 所示。



图 15-42　输入修改密码的运行结果

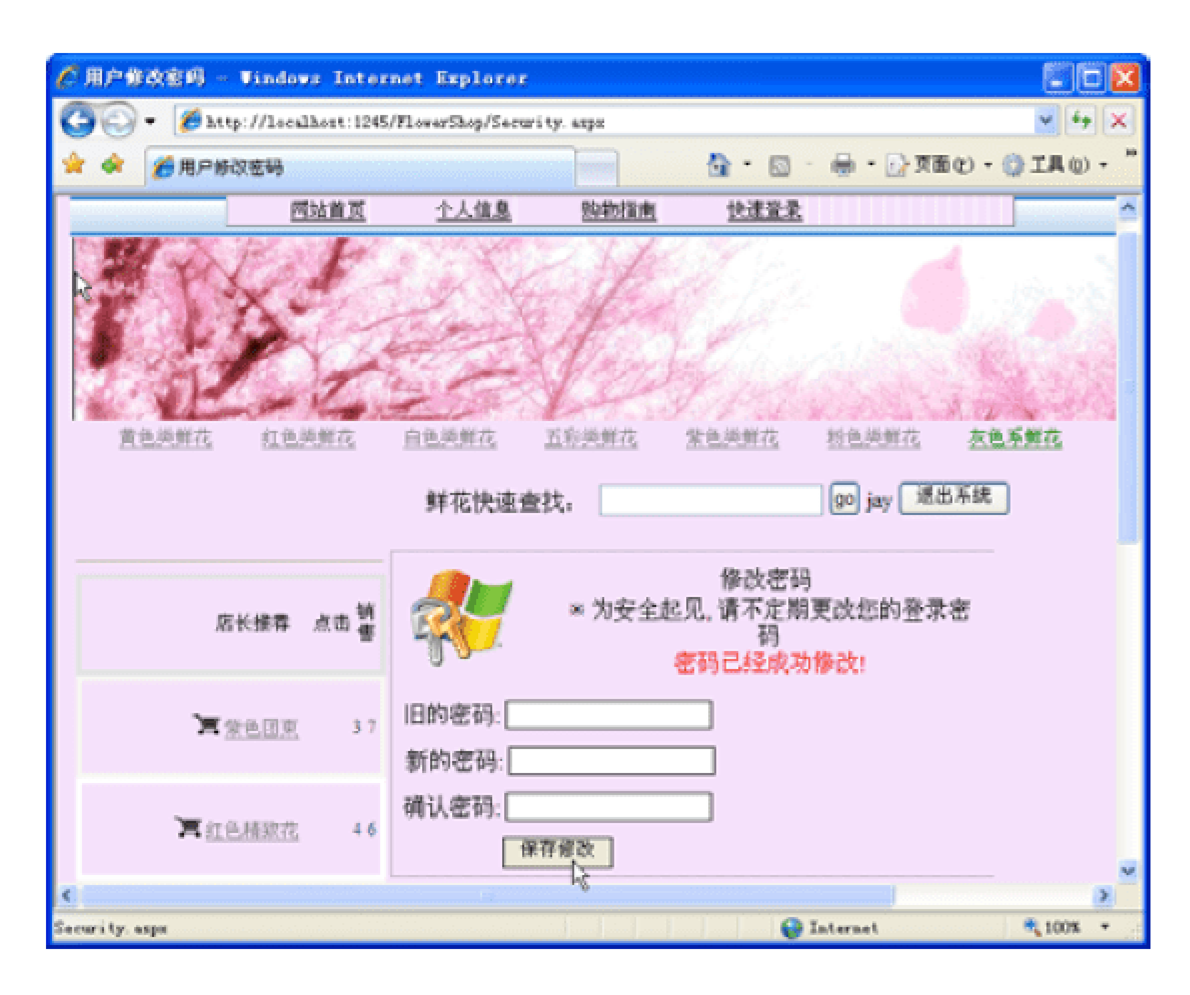


图 15-43　单击“保存修改”按钮的运行结果

15.9　总　　结

至此，一个完整的网上鲜花预订系统就完成了，一直风靡的电子商务实现也很简单吧。本例的重点就在于购物车的实现，怎样将预订的鲜花和订购人有效地联系起来，是关键问题。

此网站应用了很多用户自定义控件，是不是也发现了此技术的方便性呢，再加上母版页的技术，实现这个项目就更加得心应手了。

本例只是实现了一些基本的功能，如果读者有足够的兴趣，可以对此系统进行拓展，使之更加完善。

附录　参 考 答 案

第 1 章

一、填空题

(1)　csc.exe
(2)　80
(3)　.cs
(4)　Default.aspx

二、选择题

(1)　B
(2)　AC
(3)　A
(4)　ABC
(5)　D

第 2 章

一、填空题

(1)　GET
(2)　ErrorMessage、ErrorMessage、Text
(3)　主体
(4)　.ascx、UserControl
(5)　TreeView、SiteMapPath

二、选择题

(1)　ABC
(2)　A
(3)　A
(4)　C
(5)　D

第 3 章

一、填空题

(1)　GET
(2)　AutoEventWirup
(3)　text/xml
(4)　Server
(5)　Write
(6)　

二、选择题

(1)　C
(2)　C
(3)　D
(4)　D
(5)　B

第 4 章

一、填空题

(1)　0
(2)　System.Collections.Generic
(3)　DateTime.Now.ToString()
(4)　Insert()
(5)　引用类型、引用类型

二、选择题

(1)　ABCD
(2)　C
(3)　B
(4)　D
(5)　AD

第 5 章

一、填空题

(1) Connection
(2) Close
(3) StoredProcedure
(4) object
(5) System.IO
(6) reader.Read()

二、选择题

(1) B
(2) C
(3) ABD
(4) B
(5) A
(6) D

第 6 章

一、填空题

(1) 6
(2) DataSourceID
(3) TemplateField
(4) Bind
(5) Repeater
(6) DetailsView

二、选择题

(1) C
(2) C
(3) A
(4) B
(5) ABD

第 7 章

一、填空题

(1) XMLHttpRequest 对象
(2) 0、4
(3) var xmlDom=new ActiveXObject("Microsoft.XmlDom")、load()方法
(4) "/Server.aspx?t=" +Math.random()、onreadystatechange、str
(5) ScriptManager、ScriptManagerProxy

二、选择题

(1) C
(2) DBAC
(3) A
(4) A
(5) A
(6) D

第 8 章

一、填空题

(1) System.Web.Mvc.Controller
(2) 控制器
(3) /News/List
(4) OnException
(5) [MyProj.Filter. LogFilter]

二、选择题

(1) B
(2) A
(3) D
(4) C
(5) D
(6) B

第 9 章

一、填空题

(1) LoadVars
(2) navigateToURL()
(3) XML.firstChild、null
(4) RemoteObject
(5) ResultEvent.RESULT
(6) Flash.net

二、选择题

(1) C
(2) AD
(3) B
(4) A
(5) D
(6) B

第 10 章

一、填空题

(1) Graphics 类
(2) DashStyle
(3) Point、Rectangle
(4) FromFile
(5) Font(“宋体”, 20)、Color.Red、addText

二、选择题

(1) C
(2) D
(3) D
(4) C
(5) B
(6) A
(7) B

第 11 章

一、填空题

(1) Exists

(2) FileInfo

(3) Directory.GetCurrentDirectory()、"E:\\C#\\2010"或@"E:\C#\2010"、SetCurrentDirectory("E:\\C#\\2010");

(4) 写入、StreamReader

(5) Registry、Microsoft.Win32、密封

二、选择题

(1) A

(2) CD

(3) A

(4) D

(5) ABD

(6) AB

第 12 篇

一、填空题

(1) 对称加密

(2) 19

(3) MyHandler.PNGHandler

(4) Microsoft.Win32.Registry

(5) HttpModuler

(6) Assert

二、选择题

(1) B

(2) D

(3) B

(4) C

(5) ABCD

(6) A

(7) B

第 13 章

一、填空题

(1) lblTitle、测试、pnlMain.Controls.Add(lbl)

(2) domain.Query("ayhncn.com")

(3) SmtpClient

(4) From、To

(5) itzcn、000000、sc.Send(MailM)

二、选择题

(1) A

(2) C

(3) C

(4) D

(5) A